DOCUMENT RESUME

ED 344 030	CE 060 779
TITLE	A Secondary/Post-Secondary Curriculum for the ADA Programming Language.
INSTITUTION	Marion County Schools, Fairmont, W. Va.
SPONS AGENCY	Office of Vocational and Adult Education (ED), Washington, DC.
PUB DATE	91
CONTRACT	V199A00056
NOTE	205p.; For related documents, see CE 060 777-778.
PUB TYPE	Guides - Classroom Use - Teaching Guides (For Teacher) (052)
EDRS PRICE	MF01/PC09 Plus Postage.
DESCRIPTORS	Behavioral Objectives; Computer Assisted Instruction; Computer Science Education; Computer Software; Curriculum Guides; Instructional Materials; *Laboratory Experiments; Learning Activities; Lesson Plans; Postsecondary Education; *Programers; *Programing; *Programing Languages; Secondary Education; Technical Education; Vocational Education
IDENTIFIERS	*Ada (Programing Language)

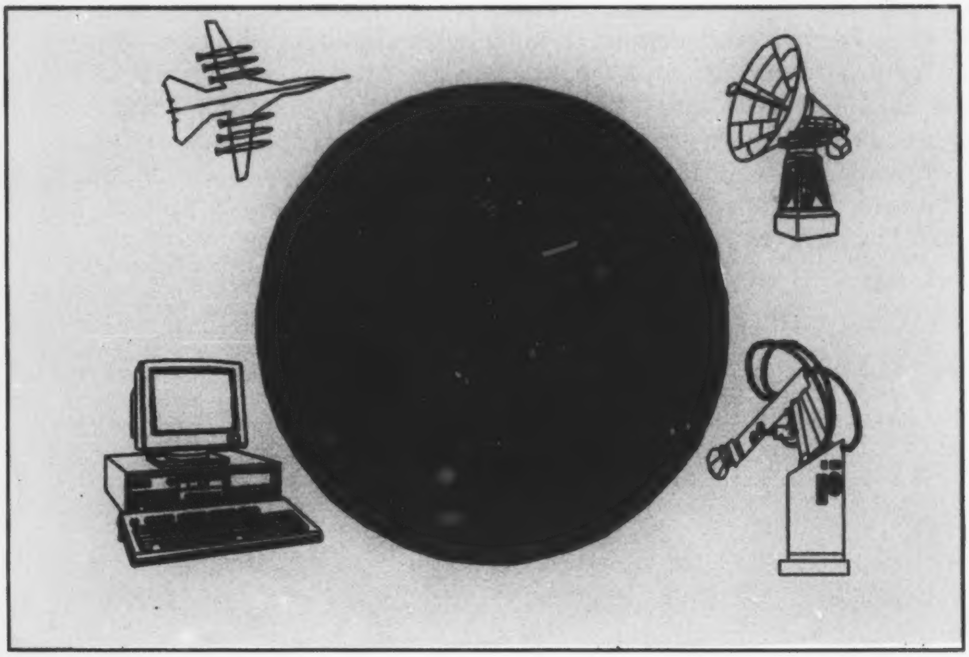
ABSTRACT

This guide provides materials for a three-block curriculum to teach the Ada computer programming language to secondary and postsecondary vocational students. The curriculum design and strategy has the following components: lectures, audiovisual aids, computer-aided instructional training and reference materials, laboratory experiences, and feedback devices. Block I, "Ada and the Department of Defense," consists of five units and four labs. Block II, "Fundamentals of Ada Programming," contains 19 units and 20 labs. In Block III, "Advanced Ada Topics," there are 11 units and 8 labs. Laboratory experiments consist of student worksheets and teacher guides. Each provides some or all of these components: block, unit, lab number, lab title; student objectives; procedure; questions; and a list of any required materials. In addition, the teacher guide provides teacher notes as needed. The information lesson plan for each unit consists of the following components: block, unit, and lesson title; lesson objectives; learning activities; special resource list; presentation (which includes an introduction and instructional topics and key points); and questions with answer key. (YLB)

****
*   Reproductions supplied by EDRS are the best that can be made   *
*   from the original document.   *
****

ED344030

A SECONDARY/POST-SECONDARY CURRICULUM FOR THE ADA® PROGRAMMING LANGUAGE



U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

☑ This document has been reproduced as received from the person or organization originating it.

☐ Minor changes have been made to improve reproduction quality

- Points of view or opinions stated in this document do not necessarily represent official OERI position or policy

Developed by the Marion County Technical Center, Farmington, West Virginia for the Robotics/Automation Technology Program under Federal Cooperative Demonstration PR/Award Number V199A00056.

CE 060 779

BEST COPY AVAILABLE

The program solutions included in this curriculum have been included for their instructional value. They have been tested with care, but are not guaranteed. In many cases, string output statements to the monitor have been sliced to two lines for better readability, and must be concatenated in order to compile properly. The Marion County Technical Center's Robotics/Automation Technology program does not offer any warranties or representations, nor does it accept any liabilities with respect to this curriculum.

Ada® is a registerered trademark of the United States Government (Ada Joint Program Office).

IntegrAda (c), Ada Training Environment (ATE) (c), "On-Line" Training and Reference Module (c), and AdaUser Libraries (c) are copyrighted trademarks of AETECH, Inc., 380 Stevens Avenue, Suite 212, Solana Beach, CA 92075.

Table of Contents

Page

Forward
Introduction 1
Curriculum Design and Strategy 1
Method of Teaching 4

BLOCK I -- Ada and the Department of Defense

Lab 01 -- Introduction to Computer Assisted Instruction Software 6
Lab 02 -- Introduction to the AETECH's *"IntegrAda"* with *"On-Line Training and Reference Module"* 10
Unit A -- The Software Crisis 12
Unit B -- Goals for Software 17
Lab 03 -- "MountainNet / AdaNet Demonstration" . . . 21
Unit C -- Software Engineering 24
Unit D -- A Brief History of the Ada Programming Language 27
Lab 04 -- Ada Information Clearinghouse 30
Unit E -- Defense Directives 35

BLOCK II -- Fundamentals of Ada Programming

Unit A -- A Basic Ada Program 37
Lab 05 -- A Basic Ada Program 42
Unit B -- Existing Packages 44
Lab 06 -- Existing Packages 46
Unit C -- Package Text_IO 49
Lab 07 -- Ohm's Law 52
Unit D -- Package Standard 56
Lab 08 -- Working with Package Standard 60
Unit E -- Simple Declarations 62
Lab 09 -- Simple Declarations Worksheet 65
Unit F -- Using Existing Packages; Parameters, Specifications, and Calls . . . 68
Lab 10 -- Combining Existing Packages 71
Unit G -- Data Types 73
Unit H -- Ada Scalar Types 75
Lab 11 -- Scalar Types 77
Unit I -- Enumeration Types 79
Lab 12 -- Enumeration Types 81
Unit J -- Derived Types 83
Lab 13 -- Derived Types 85

Table of Contents
(Continued)

Page

Unit K -- Subtypes 87
Lab 14 -- Subtypes 89
Unit L -- Subprograms 92
Lab 15 -- Subprograms 94
Unit M -- Packages 97
Lab 16 -- Creating Simple Packages 100
Unit N -- Declaring Subprograms and Creating Packages 103
Lab 17 -- Declaring Subprograms and Creating Packages 105
Unit O -- Ada Language Syntax 110
Lab 18 -- Ada Language Syntax: Using Comments 113
Unit P -- The 'If' Control Structure 115
Lab 19 -- The If..Then Control Structure 117
Lab 20 -- The If..Then..Elsif..Else Control Structure 119
Unit Q -- The Case Control Structure 121
Lab 21 -- The Case Control Structure 123
Unit R -- The Loop Control Structure 125
Lab 22 -- The Loop Control Structure 127
Lab 23 -- Loop and Reverse Loop 129
Lab 24 -- The While..Loop Control Structure 132
Unit S -- Style 134

BLOCK III -- Advanced Ada Topics

Unit A -- Type Attributes 137
Unit B -- More Attributes 139
Unit C -- Records 141
Lab 25 -- Records 143
Unit D -- Arrays 146
Lab 26 -- Arrays 149
Unit E -- Exceptions 152
Lab 27 -- Exceptions 157
Unit F -- Private Types 160
Unit G -- Generics 163
Lab 28 -- Generics 166
Unit H -- Sequential Files 169
Lab 29 -- Sequential Files 172
Unit I -- Direct Access Files 178
Lab 30 -- Direct Access Files 181
Unit J -- Introduction to Tasks 185
Lab 31 -- Introduction to Tasks 188
Unit K -- Tasks and Task Communication 190
Lab 32 -- Task Communication 194

FORWARD

This curriculum was designed implementing the IntegrAda environment and CAI module available from the AETECH Corporation. These products were chosen from other Ada development products because of:

1) The ease of programming in this environment;

2) The ability to utilize IBM/IBM compatible personal computer platforms;

3) The built in packages such as SOUND, and GRAPHICS which suit themselves nicely to the Ada classroom, creating an enjoyable learning environment;

4) The ability to have a Computer Aided Instructional Package;

5) The ability to utilize existing IBM/PC compatible equipment without requiring the purchase of math co-processors, extra memory, or new computers; and

6) The ability to use the standard Ada compiler, CAI program, environment, and other tools in common use today throughout the military and Department of Defense.

The IntegrAda environment is the standard Ada development environment utilized by:

a. U.S. Air Force - DESKTOP III contract Ada compiler and tools (The DESKTOP III contract will distribute over 250,000 PW2 386 PC's to all branches of the Department of Defense, and other government agencies).

b. U.S. Navy - Naval Postgraduate School, Montery, CA; used to teach Ada and Software Engineering classes.

c. U.S. Marine Corps - Quantico, VA; used in daily software development.

d. U.S. Army - CECOM, Ft. Monmouth, NJ; used for the development of Command and Control Systems for the Army.

e. and many others.

We wish to thank the AETECH Corporation for their support in the development of this curriculum.

A SECONDARY/POST-SECONDARY CURRICULUM FOR "THE ADA PROGRAMMING LANGUAGE"

INTRODUCTION

Perhaps no other computer language has created quite as much excitement as the Ada computer programming language. Developed under a contract for the Department of Defense, Ada has become the programming language of the future. Since its inception in the early 1980's, Ada has grown from being only a language used and proliferated in the defense community, to a language which businesses and educational institutions have come to use. Because of its structure, Ada is an excellent language to use to write programs. However, due to the large amount of Ada code to be generated, we now face an extreme shortage of trained Ada programmers. With this lack of trained programmers in mind, this secondary/post secondary curriculum has been developed to teach Ada to students, in hopes of meeting the demand for a trained Ada community.

CURRICULUM DESIGN AND STRATEGY

The curriculum employs a strategy which includes, but is not limited to, the following components: lectures, audio/visual aids, computer aided instructional training and reference materials, laboratory experiences, and feedback devices. The entire curriculum is available in hardcopy form through the West Virginia Curriculum Repository located at Cedar Lakes, West Virginia. The address of the West Virginia Curriculum Repository is:

Curriculum Technology Resource Center
Cedar Lakes Conference Center
Ripley, West Virginia 25271
(304) 372-7021

A. Lectures - Lectures will be provided by the individual instructor, using the included informational lesson plans and curriculum as guides. The individual instructor should provide lecture notes as required.

B. Audio/ Visual Aids - Audio/ visual aids should be utilized to supplement this curriculum. It is felt that each individual instructor wishing to utilize the curriculum will have access to an overhead projector and a VHS video playback machine. Many audio/visual aids, utilized for the construction of this curriculum, are available through the West Virginia Curriculum Repository at Cedar Lakes, West Virginia for dissemination to interested instructors.

Computer Aided Instructional and Training Reference Materials - This curriculum was designed utilizing the "Ada Training Environment" and "IntegrAda" with optional "On-Line Training and Reference Module". These are commercial Computer Aided Instruction (CAI) programs available from the AETECH Corporation for IBM PC compatible computers. Studies have shown that students will read and comprehend information at a faster rate if it is presented interactively on a computer terminal, rather than in a textbook. The software chosen for use with this project was developed over a five year period and field-tested by the AETECH Corporation. It is not meant as a replacement for individual instructor lectures or demonstrations; but it can considerably enhance the learning process when used with this curriculum.

Laboratory Experiences - It is felt that in order for a student to gain proficiency with the Ada language, laboratory experiences should be provided to contribute to the student's overall learning. Many laboratory experiences are included in this curriculum, which will allow the student to demonstrate, through the use of computer programming exercises, their proficiency with the language. It is felt that a computer to student ratio is 1:1 is needed.

The laboratory exercises in this curriculum were designed around a one hour format using IBM PC compatible computers and "IntegrAda", the validated Ada compiler for the IBM PC compatible produced by AETECH, Inc. Each laboratory exercise should be preceded by lecture, audio/visual instruction, and CAI instruction where appropriate.

Feedback Devices - Feedback devices are provided within Block I of this curriculum. It is felt that instructors generally prefer to develop their own quizzes and tests, and no feedback devices have been included in Blocks II and III. Pre-enrollment and post-enrollment attitude measurement devices should be administered by the Instructor, to be utilized as tools for measuring students' attitudes toward computers in general, and the Ada programming language in particular.

METHOD OF TEACHING - It is felt that, in order for the curriculum to be effective, the following teaching method should be incorporated as a strategic guide, to insure that the curriculum is effective:

1. Teacher Lecture - The teacher will present the required lecture materials to the students, who will in turn take notes on the presented material. Lectures shall include audio/visual tools as required.

2. Student Participation - After each presented lesson by the teacher, all students should be given an opportunity to ask questions, express concerns, or make comments concerning the presented material.

3. Computer Aided Instruction - It is recommended that a CAI package be incorporated as part of the total Ada curriculum. This CAI package should be made available for students to view after the presented lecture material, and prior to any laboratory experience.

4. Laboratory Experience - As much as possible, laboratory experiences should follow the presented lecture material and CAI training. Laboratory experiences should include actual programming tasks, and to simplify the learning process at the secondary level, it is highly recommended that Ada programming tools used in the laboratory include user-friendly "Turbo-like" Ada programming systems with simple, easy-to-use libraries for Screen, Mouse, Sound, and Pixels.

5. Feedback Devices - Feedback devices should be utilized after each lecture or completed laboratory experience as required. The individual teacher will have the better idea of when feedback from students is required. It is crucial that feedback devices for measuring students attitudes prior to beginning the curriculum, and feedback devices for measuring students' attitudes after the completion of the course, should be administered at the appropriate times.

LABORATORY EXPERIMENT

I. **BLOCK:** I

II. **UNIT:** Introduction

III. **LAB NUMBER:** 01

IV. **LAB TITLE:** "Introduction to Computer Assisted Instruction Software"

V. **STUDENT OBJECTIVES:** At the completion of this experiment, the student should be able to:

1. Boot up the AETECH Ada Training Environment or the AETECH On-Line Training and Reference Module used for this curriculum.
2. Choose from the displayed menus which lesson they would like to study.
3. Return to the operating system from the CAI package.

VI. **REQUIRED MATERIALS:** AETECH *"Ada Training Environment"* or AETECH *"IntegrAda"* with the *"On-Line Training and Reference Module"*.

VII. **PROCEDURE**

1. Power on.
2. Log on to the drive where the *"Ada Training Environment"* or *"IntegrAda"* with the *"On-Line Training and Reference Module"* are installed.
3. At the DOS prompt, change to the working directory where the CAI package has been installed. If the *"Ada Training Environment"* has been installed, then type CD\ATE\COURSE. If the On-Line Training and Reference Module has been installed, then type CD\IADA.
4. At the DOS prompt, type the appropriate command for the CAI package which has been installed. If the *"Ada Training Environment"* has been installed, then type ATE. If the On-Line Training and Reference Module has been installed, then type REFER.

5. In the lower right corner of the screen, a square labeled "Ada Training Environment" will appear. Each tutorial block is listed. Select the proper block by using the up/down cursor key until the correct block is highlighted, then press Enter.

6. Within each block, there is a series of lessons. Unless otherwise instructed, you will take the lessons in order. Select the proper lesson by using the up/down cursor key, then press Enter.

7. Within each lesson, there is a list of topics. Most topics consist of one screen of information. Select each topic in order by using the up/down cursor keys, then pressing Enter.

8. Read each screen of information, and take notes on the key points presented.

9. Press Enter. This will take you back to the list of topics.

10. When you have completed the assignment, select QUIT from the topic menu. This will return you to the DOS prompt.

11. Practice returning to the operating system and booting up the CAI system several times so that you are extremely familiar with this procedure.

12. Power down computer, and clean up area.

13. Record any questions, comments, or concerns you may have with using the system for your Instructor.

TEACHER GUIDE
LABORATORY EXPERIMENT

I. **BLOCK:** I ***(Teacher Note: This laboratory is only applicable to those instructors who are utilizing the AETECH "Ada Training Environment" or the AETECH "IntegrAda" with "On-Line Training and Reference Module").***

II. **UNIT:** Introduction

III. **LAB NUMBER:** 01

IV. **LAB TITLE:** "Introduction to Computer Assisted Instruction Software"

V. **STUDENT OBJECTIVES:** At the completion of this experiment, the student should be able to:

1. Boot up the AETECH Ada Training Environment or the AETECH On-Line Training and Reference Module used for this curriculum.
2. Choose from the displayed menus which lesson they would like to study.
3. Return to the operating system from the CAI package.

VI. **REQUIRED MATERIALS:** AETECH *"Ada Training Environment"* or AETECH *"IntegrAda"* with the *"On-Line Training and Reference Module"*.

VII. **PROCEDURE**

1. Power on.
2. Log on to the drive where the *"Ada Training Environment"* or *"IntegrAda"* with the *"On-Line Training and Reference Module"* are installed.
3. At the DOS prompt, change to the working directory where the CAI package has been installed. If the "Ada Training Environment" has been installed, then type CD\ATE\COURSE. If the On-Line Training and Reference Module has been installed, then type CD\IADA.

4. At the DOS prompt, type the appropriate command for the CAI package which has been installed. If the "Ada Training Environment" has been installed, then type ATE. If the On-Line Training and Reference Module has been installed, then type REFER.
5. In the lower right corner of the screen, a square labeled "Ada Training Environment" will appear. Each tutorial block is listed. Select the proper block by using the up/down cursor key until the correct block is highlighted, then press Enter.
6. Within each block, there is a series of lessons. Unless otherwise instructed, you will take the lessons in order. Select the proper lesson by using the up/down cursor key, then press Enter.
7. Within each lesson, there is a list of topics. Most topics consist of one screen of information. Select each topic in order by using the up/down cursor keys, then pressing Enter.
8. Read each screen of information, and take notes on the key points presented.
9. Press Enter. This will take you back to the list of topics.
10. When you have completed the assignment, select QUIT from the topic menu. This will return you to the DOS prompt.
11. Practice returning to the operating system and booting up the CAI system several times so that you are extremely familiar with this procedure.
12. Power down computer, and clean up area.
13. Record any questions, comments, or concerns you may have with using the system for your Instructor.

LABORATORY EXPERIMENT

I. **BLOCK:** I

II. **UNIT:** Introduction

III. **LAB NUMBER:** 02

IV. **LAB TITLE:** Introduction to the AETECH's "IntegrAda" with "On-Line Training and Reference Module"

V. **STUDENT OBJECTIVES:** At the completion of this experiment, the student should be able to:

1. Follow the oral instructions given by your Instructor on entering the IntegrAda environment.
2. Follow the written instructions within the IntegrAda Reference Manual, Chapter 12, "Getting Started", and gain an understanding of how to create, edit, compile, bind, execute, and print Ada programs.

VI. **REQUIRED MATERIALS:**

1. Note taking materials.
2. *"IntegrAda"* with *"On-Line Training and Reference Module"*.
3. IntegrAda Reference Manual, Chapter 12: *"Introductory Session"*.

VII. **PROCEDURE**

1. Follow the procedures outlined in Chapter 12 of the IntegrAda Reference Manual, pages 12-2 through 12-30. Since the system has already been installed for you, follow the oral instructions given to you by your Instructor on entering the system. Continue at step 5, page 12-2 of the IntegrAda Reference Manual.
2. Record any questions you have about using the editor environment.
3. Power down computer, and clean up area.

TEACHER GUIDE
LABORATORY EXPERIMENT

I. **BLOCK:** I *(Teacher Note: This laboratory is only applicable to those instructors who are utilizing the AETECH "IntegrAda" with "On-Line Training and Reference Module".)*

II. **UNIT:** Introduction

III. **LAB NUMBER:** 02

IV. **LAB TITLE:** Introduction to the AETECH's "IntegrAda" with "On-Line Training and Reference Module"

V. **STUDENT OBJECTIVES:** At the completion of this experiment, the student should be able to:

1. Follow the oral instructions given by your Instructor on entering the IntegrAda environment.
2. Follow the written instructions within the IntegrAda Reference Manual, Chapter 12, "Getting Started", and gain an understanding of how to create, edit, compile, bind, execute, and print Ada programs.

VI. **REQUIRED MATERIALS:**

1. Note taking materials.
2. *"IntegrAda"* with *"On-Line Training and Reference Module"*.
3. IntegrAda Reference Manual, Chapter 12: *"Introductory Session"*.

VII. **PROCEDURE**

1. Follow the procedures outlined in Chapter 12 of the IntegrAda Reference Manual, pages 12-2 through 12-30. Since the system has already been installed for you, follow the oral instructions given to you by your Instructor on entering the system. Continue at step 5, page 12-2 of the IntegrAda Reference Manual.
2. Record any questions you have about using the editor environment.
3. Power down computer, and clean up area.

BLOCK I

BLOCK I

Ada and the Department of Defense

INFORMATION LESSON PLAN

I. **BLOCK:** I - " Ada and the Department of Defense"

II. **UNIT:** A

III. **LESSON TITLE:** "The Software Crisis"

IV. **LESSON OBJECTIVES:** At the completion of this lesson, the student should be able to :

1. Attribute advances in computer hardware and software, in the late sixties and early seventies, with the advent of the microprocessor.
2. Identify amount of DOD annual expenditures for software.
3. Identify two factors responsible for increased cost of software.
4. Define "Software Maintenance".
5. List the six problem areas associated with software development.

V. **LEARNING ACTIVITIES:**

1. Take notes on lecture presented by Instructor.
2. Participate in class discussion of presented lecture.
3. CAI Assignment - Block I, Unit 1
   AETECH *"Ada Training Environment"* or *"IntegrAda"* with *"On-Line Training and Reference Module"*.

Read & take notes on following sections:

a. Increasing demand for software.
b. Increasing software costs.
c. Software to hardware costs.
d. Software maintenance costs.
e. Other major software costs.
f. Problems with software development.
g. Other life cycle problems.
h. Problems with quality.
i. Language proliferation.

VI. SPECIAL RESOURCES:

AETECH *"Ada Training Environment"* and *"IntegrAda"* with *"On-Line Training and Reference Module"*.

Olson/Whitehall, Ada For Programmers, Ch. 1.

Stein, Ada: A Life and Legacy, Preface.

Engle/Dominice, Introductory Ada Workshop.

Softech, Basic Ada Programming L202- U.S. Army, Vol I.

Sommerville/Morrison, Software Development with Ada Ch. 1.

VII. PRESENTATION:

A. Introduction

1. Tell "Invasion of Grenada" story where Army's and Navy's computers couldn't communicate with each other, requiring an Army officer to make a credit card long distance call to North Carolina for naval support.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
1. Increase Demand for Software	1a. Demands due to microprocessor which made systems more efficient, reliable, and accurate.
2. Increased Cost for Software	2a. 6 billion annually by DOD.
3. Factors responsible for software costs	3a. Technological advances.
	3b. Higher salaries as demands for highly skilled programmers exceeds supply.
4. Software Maintenance	4a. Definition - Program which now works that must be changed or modified to work differently.

TOPIC	KEY POINT
5. Problems with Software Development	5a. Unmodifiable - No one other than writer(s) can interpret software.
	5b. Nontransportable- Software written & tailored to specific machine, and does not work on another machine.
	5c. Not Timely - Typically, software systems are delivered late.
	5d. Unresponsive - Software doesn't perform as required.
	5e. Inefficient - Software is frequently larger & slower than anticipated.
	5f. Unreliable - Software typically fails.

"The Software Crisis"

QUESTIONS

Place your answer in the provided space for each of the following questions.

1. Name two factors which are responsible for the high cost of software.

2. Define Software Maintenance.

3. Name the single factor which provided an enormous demand for software.

4. List five of the six problems associated with software development.

   a.

   b.

   c.

   d.

   e.

5. How much does the U.S. Department of Defense spend annually for software?

"The Software Crisis

ANSWERS TO QUESTIONS

1. Name two factors which are responsible for the high cost of software.

   Technological advances, demands for skilled programmers

2. Define Software Maintenance.

   Operation/ program which now works but must be changed or modified to work differently.

3. Name the single factor which provided an enormous demand for software.

   Microprocessor

4. List five of the six problems associated with software development.

   a. **Unmodifiable**

   b. **Nontransportable**

   d. **Not Timely**

   e. **Inefficient**

   f. **Unreliable**

   g. **Unresponsive**

5. How much does the U.S. Department of Defense spend annually for software?

   U.S. DOD spends approximately 6 billion dollars annually for software.

INFORMATION LESSON PLAN

I. **BLOCK:** I - "Ada and the Department of Defense"

II. **UNIT:** B

III. **LESSON TITLE:** "Goals for Software"

IV. **LESSON OBJECTIVES:** At the completion of this lesson, the student should be able to :

1. List the four goals for software.
2. Describe what understandable software is.
3. Define Modifiability as related to software.
4. Discuss the impact of reliable versus unreliable software systems.
5. Define Efficiency as related to software.
6. Discuss the importance of software Portability.

V. **LEARNING ACTIVITIES:**

1. Take notes on lecture presented by Instructor.
2. Participate in class discussion of presented lecture.
3. CAI Assignment - Block I, Unit 2
   AETECH *"Ada Training Environment"* or *"IntegrAda"* with *"On-Line Training and Reference Module"*.

   Read & take notes on following sections:

   a. Understandability.
   b. Modifiability.
   c. Reliability.
   d. Efficiency.
   e. Portability.

VI. **SPECIAL RESOURCES:**

AETECH *"Ada Training Environment"* and *"IntegrAda"* with *"On-Line Training and Reference Module"*.

Engle/Dominice, Introductory Ada Workshop.

Softech, Basic Ada Programming L202-U.S. Army, Vol I.

VII. PRESENTATION:

A. Introduction

1. E.G. Booth quote:
   "The basic problem is not our mismanagement of technology, but rather our inability to manage the complexity of our systems".

B. Instructional Topics and Key Points

TOPIC	KEY POINT
1. Five Goals of Software	1a. Understandability - software must be understood by anyone who will write the code, look at the code, or modify the code. Software may only be written one time, but it is read many times. The easier software is to read, the easier software is to understand, the easier it is modify. (Refer back to costs for maintaining software).
	1b. Modifiability - Allows program to be changed to meet the new needs of the user, with a minimum of time and expense.
	1c. Reliability - Software must perform as it is supposed to (Nuclear Attack Warning System Example). DOD reliability factors: a. Systems are lethal. b. Many systems are unattended. c. Systems must be fault-tolerant.
	1d. Efficiency - Achieving maximum performance within small hardware constraints. There are two ways to measure efficiencies of software systems: a. Amount of code. b. Speed of execution.
	1e. Portability - Ability of a program to be used on different computers, where the software is not hardware dependent.

"Goals for Software"

QUESTIONS

Place your answer in the provided space for each of the following questions.

1. Define Understandability.

2. Define Modifiability.

3. Define Reliability.

4. Define Efficiency.

5. Define Portability.

"Goals for Software"

ANSWERS TO QUESTIONS

1. Define Understandability.

   Understandability - Software must be understood by anyone who will write, read, or modify the code.

2. Define Modifiability.

   Modifiability - Allows program to be changed to meet new requirements without having to write a new program.

3. Define Reliability.

   Reliability - Software must perform as it is supposed to.

4. Define Efficiency.

   Efficiency - Achieving maximum performance, 2 ways to measure by a. small size and b. high speed.

5. Define Portability.

   Portability - Ability of a program to be transported from one computer to another (software is not hardware dependent).

LABORATORY EXPERIMENT

I. **BLOCK:** I - "Ada and the Department of Defense"

II. **UNIT:** B "Goals for Software"

III. **LAB NUMBER:** 03

IV. **LAB TITLE:** "MountainNet / AdaNet Demonstration"

V. **STUDENT OBJECTIVES:** At the completion of this experiment, the student should be able to:

1. Understand the services provided by MountainNet and the AdaNet Bulletin Board.
2. Access AdaNet.
3. Sign on and download a file or bulletin from Adanet.
4. Answer the questions at the end of this experiment.

VI. **REQUIRED MATERIALS:**

1. Note taking materials.
2. Blank registration forms for AdaNet.
3. A blank formatted disk.

VII. **PROCEDURE**

Follow the oral instructions for accessing AdaNet. Get into the system, and explore various topics, drawers, etc. Choose various materials that you would like to keep, and download these files onto your blank formatted disk.

VIII. **Questions**

1. What is MountainNet? What is AdaNet?
2. What kinds of information are available from AdaNet?

TEACHER GUIDE
LABORATORY EXPERIMENT

I. **BLOCK:** I - "Ada and the Department of Defense"

II. **UNIT:** B "Goals for Software"

III. **LAB NUMBER:** 03

IV. **LAB TITLE:** "MountainNet / AdaNet Demonstration"

V. **STUDENT OBJECTIVES:** At the completion of this experiment, the student should be able to:

1. Understand the services provided by MountainNet and the AdaNet Bulletin Board.
2. Access AdaNet.
3. Sign on and download a file or bulletin from AdaNet.
4. Answer the questions at the end of this experiment.

VI. **REQUIRED MATERIALS:**

1. Note taking materials.
2. Blank registration forms for AdaNet.
3. A blank formatted disk.

VII. **PROCEDURE**

TEACHER NOTE: Contact the MountainNet User Representative well in advance of the desired date for the demonstration. If a representative is unable to come to your school for a demonstration, it is recommended that the instructor be familiar enough with the AdaNet system to provide the demonstration. Otherwise, request registration forms so that the registration may be completed and the students already have their packets of information for accessing AdaNet before the demonstration is presented. The address of MountainNet is:

MountainNet, Inc.
Eastgate Plaza
P.O. Box 370
Dellslow, WV 26531-0370
(800) 444-1458

Follow the oral instructions for accessing AdaNet. Get into the system, and explore various topics, drawers, etc. Choose various materials that you would like to keep, and download these files onto your blank formatted disk.

VIII. Questions

1. What is MountainNet? What is AdaNet?

   MountainNet is a telecommunications corporation in Dellslow, WV, whose purpose is to run the AdaNet system. AdaNet is an information service and software reuse research project designed to provide public domain software engineering and Ada for business, government, and academe.

2. What kinds of information are available from AdaNet?

   Ada source code libraries, bibliographic references and publication information, descriptions of public and commercial repositories, directories of products, listings of organizations, listing of forums, etc.

INFORMATION LESSON PLAN

I. **BLOCK:** I - "Ada and the Department of Defense"

II. **UNIT:** C

III. **LESSON TITLE:** "Software Engineering"

IV. **LESSON OBJECTIVES:** At the completion of this lesson, the student should be able to :

1. Define Software Engineering.
2. Describe Abstraction.
3. Define Modularity.
4. Describe Localization.
5. Understand the principle of information hiding.
6. Understand the principle of completeness.
7. Define Confirmability.

V. **LEARNING ACTIVITIES:**

1. Take notes on lecture presented by Instructor.
2. Participate in class discussion of presented lecture.
3. CAI Assignment - Block I, Unit 3
   AETECH *"Ada Training Environment"* or *"IntegrAda"* with *"On-Line Training and Reference Module"*.

   Read & take notes on following sections:

   a. Combat Logistics Support Example.
   b. Abstraction.
   c. Modularity.
   d. Localization.
   e. Information Hiding.
   f. Completeness.
   g. Confirmability.

VI. SPECIAL RESOURCES:

AETECH *"Ada Training Environment"* and *"IntegrAda"* with *"On-Line Training and Reference Module"*.

Engle/Dominice, Introductory Ada Workshop.

Softech, Basic Ada Programming L202- U.S. Army, Vol I.

VII. PRESENTATION:

A. Introduction

1. Tell students that there is no uniform concensus for the definition of Software Engineering; then tell them that the following are goals of Software Engineering for DOD.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
1. Abstraction	1a. As used in program development, is a process in which a system is viewed at several levels from simple to complex (top-down approach); where the programmer concentrates on the essentials, leaving the details for later time.
2. Modularity	2a. Programming tasks may be broken into individual modules (divide and conquer). 2b. Module - Unit of code or program which may be written, tested, and function independently of other modules.
3. Localization	3a. Related pieces of program code should be found in the system at close proximity to one another. It would not be wise to put two pieces of related code in separate modules.
4. Information Hiding	4a. Programmer writes parts of system which are inaccessible to other parts of system. Makes code immune to side effects from changes, to other parts of the system, which may occur.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
5. Completeness	5a. All required components and resources for a module to function properly must be made available to that module.
6. Confirmability	6a. Software module can readily be tested with a minimum of support from other modules.

INFORMATION LESSON PLAN

I. **BLOCK:** I - "Ada and the Department of Defense"

II. **UNIT:** D

III. **LESSON TITLE:** "A Brief History of the Ada Programming Language"

IV. **LESSON OBJECTIVES:** At the completion of this lesson, the student should be able to :

1. Define "Embedded Systems".
2. Identify the company who developed the Ada programming language.
3. Gain an understanding of the history/development of the Ada programming language.
4. Define the function of the AJPO.
5. Discuss the naming of the language.

V. **LEARNING ACTIVITIES:**

1. Take notes on lecture presented by Instructor.
2. Participate in class discussion of presented lecture.
3. CAI Assignment - Block I, Unit 4
   AETECH *"Ada Training Environment"* or *"IntegrAda"* with *"On-Line Training and Reference Module"*.

   Read & take notes on following sections:

   a. A language for embedded computers.
   b. The Higher Order Language Working Group.
   c. Establishing the requirement.
   d. Starting the design.
   e. Completion of the design effort.
   f. Naming the new language.
   g. Ada Joint Program Office.

VI. **SPECIAL RESOURCES:**

AETECH *"Ada Training Environment"* and *"IntegrAda"* with *"On-Line Training and Reference Module"*.

VII. PRESENTATION:

A. Introduction

1. Give a brief oral history of why DOD needed a Higher Order Language which could meet their needs.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
1. Embedded Systems	1a. Definition - a possible group of machines that are controlled by one or more computers, and function as one independent unit.
2. Development of Ada	2a. 1975 - Higher Order Language Working Group (HOLWG) Made up of members from academe, government, industry, and the three branches of the military, whose purpose was to review existing computer programming languages, and to develop requirements of the new computer language for use with DOD projects. 1. Specify requirements of a language. 2. Evaluate current languages against DOD requirements. 3. Make recommendations on a language to use based on that evaluation, or possibly recommend the creation of a new language. 2b. 1975 - Strawman document was developed; document provided initial specifications for new language. Strawman was submitted by HOLWG for review by all parties involved in language development. Comments incorporated into Strawman led to development of Woodenman, and Tinman documents (which were changes to Strawman). Tinman returned with relatively few changes. Changes were made, and new document was called Ironman (1977).

B. Instructional Topics and Key Points

TOPIC	KEY POINT
3. Honeywell-Bull	3a. Company from Europe who won the language design competition sponsored by DOD, to develop the Ada language.
4. Naming of Language	4a. Named for Lady Augusta Ada Lovelace (born 1815); daughter of Lord Byron (poet). Worked with Charles Babbage on his analytical engine. Is considered to be first programmer (due to notes she made during work on engine.
5. AJPO	5a. Ada Joint Program Office mission is to disseminate information to military and general public concerning Ada. Runs Ada Information Clearinghouse (AdaIC) and CREASE (Catalog of Resources for Education in Ada Software). Monitors compiler compliances with DOD guidelines on Ada. Funded by AJPO through ITT, who manages the AdaIC.

LABORATORY EXPERIMENT

I. **BLOCK:** I - "Ada and the Department of Defense"

II. **UNIT:** D

III. **LAB NUMBER:** 04

IV. **LAB TITLE:** "Ada Information Clearinghouse"

V. **STUDENT OBJECTIVES:** At the completion of this experiment, the student should be able to:

1. Identify the resources available from the Ada Information Clearinghouse.
2. Write a letter requesting AdaIC to include the student on their mailing list, to receive AdaIC information.

VI. **REQUIRED MATERIALS:**

1. Note taking materials.
2. Letter bond, envelopes, and stamps.
3. Student Data Disk.

VII. **PROCEDURE**

1. Draft a letter to the Ada Information Clearinghouse, requesting to be added to their mailing list. The address is:

AdaIC
c/o ITT Research Institute
4600 Forbes Blvd., Second Floor
Lanham, Maryland 20706-4312

VIII. **QUESTIONS**

1. What is AdaIC?
2. How is the program funded?
3. What kinds of information are available from the Ada Information Bulletin Board?

TEACHER GUIDE

LABORATORY EXPERIMENT

I. **BLOCK:** I - "Ada and the Department of Defense"

II. **UNIT:** D

III. **LAB NUMBER:** 04

IV. **LAB TITLE:** "Ada Information Clearinghouse"

V. **STUDENT OBJECTIVES:** At the completion of this experiment, the student should be able to:

1. Identify the resources available from the Ada Information Clearinghouse.
2. Write a letter requesting AdaIC to include the student on their mailing list, to receive AdaIC information.

VI. **REQUIRED MATERIALS:**

1. Note taking materials.
2. Letter bond, envelopes, and stamps.
3. Student Data Disk.

VII. **PROCEDURE**

1. Draft a letter to the Ada Information Clearinghouse, requesting to be added to their mailing list. The address is:

AdaIC
c/o ITT Research Institute
4600 Forbes Blvd., Second Floor
Lanham, Maryland 20706-4312

Teacher Note: Plan a follow up session for this lab when the students receive their packets from AdaIC. They will receive information on how to access the Ada Information Bulletin Board, as well as other information that is of interest to the Ada community.

VIII. QUESTIONS

1. What is AdaIC?

   AdaIC is the Ada Information Clearinghouse which is part of the AJPO, and is designed to disseminate information to the Ada community.

2. How is the program funded?

   The program is funded by AJPO through ITT who runs the AdaIC.

3. What kinds of information are available from the Ada Information Bulletin Board?

   News articles, contract awards, validated compiler listings, training seminars, conferences, etc.

"History of Ada"

QUESTIONS

Place your answer in the provided space for each of the following questions.

1. Define Embedded Systems.

2. What does HOLWG stand for? What was it made up of? What did they do?

3. What company designed Ada?

4. How was Ada named?

5. What does AJPO stand for? What do they do?

"History of Ada"

ANSWERS TO QUESTIONS

1. Define Embedded Systems.

   Embedded Systems - A group of machinery which is controlled by one or more on-board computers, and functions as an independent unit.

2. What does HOLWG stand for? What was it made up of? What did they do?

   HOLWG - Higher Order Language Working Group, it was made up of academe, government, and military to develop the requirements for DOD's new language (Ada).

3. What company designed Ada?

   Honeywell - Bull

4. How was Ada named?

   Named for Lady Augusta Ada Lovelace, daughter of Lord Bryon. She is considered the first programmer.

5. What does AJPO stand for? What do they do?

   AJPO - Ada Joint Program Office, they operate AdaIC and CREASE and their purpose is to disseminate information about Ada, and to also oversee compliance with DOD Ada guidelines.

INFORMATION LESSON PLAN

I. **BLOCK:** I - "Ada and the Department of Defense"

II. **UNIT:** E

III. **LESSON TITLE:** "Defense Directives"

IV. **LESSON OBJECTIVES:** At the completion of this lesson, the student should be able to:

1. Discuss the differences between standardization directives and acquisition policies.
2. Understand the implications of DOD Directive 5000.31.
3. Understand the implications of DOD Directive 5000.1.

V. **LEARNING ACTIVITIES:**

1. Take notes on lecture presented by Instructor.
2. Participate in class discussion of presented lecture.
3. CAI Assignment - Block I, Unit 5
   AETECH *"Ada Training Environment"* or *"IntegrAda"* with *"On-Line Training and Reference Module"*.

   Read & take notes on following sections:

   a. Background.
   b. Warner Amendment.
   c. Higher Order Languages.
   d. Mission Critical Systems.
   e. DOD Directive 5000.31
   f. DOD Directive 5000.1
   g. DOD Directive 5000.29

VI. **SPECIAL RESOURCES:**

AETECH *"Ada Training Environment"* and *"IntegrAda"* with *"On-Line Training and Reference Module"*.

VII. PRESENTATION:

A. Introduction

1. Ask students question, "Why do you think there are rules governing the use of the Ada programming language?"

B. Instructional Topics and Key Points

TOPIC	KEY POINT
1. Standardization Directives	1a. Dictate which Higher Order Languages are allowed to be used in DOD systems.
2. Acquisition Policies	2a. Govern how systems and software are purchased by U.S. Government.
3. DOD 5000.31	3a. 1976 - Required use of an approved Higher Order Language for "Mission-Critical Systems". 3b. 1983 - stated "The Ada programming language shall become the single computer programming language for Defense Mission Critical applications."
4. DOD 5000.1	4a. Major Systems Acquisition Policy - stated "Effective Jan. 1/84 for programs entering advanced development and July 1/84 for programs entering full-scale engineering development, Ada shall be the programming language."

BLOCK II

BLOCK II

Fundamentals of Ada Programming

INFORMATION LESSON PLAN

I. **BLOCK:** II - "Fundamentals of Ada Programming"

II. **UNIT:** A

III. **LESSON TITLE:** "A Basic Ada Program"

IV. **LESSON OBJECTIVES:** At the completion of this lesson, the student should be able to:

1. Understand the purpose of each line of code in a simple Ada program.
2. Understand the concept of a package.
3. Define and identify a context clause.
4. Identify how comments are incorporated in an Ada program.
5. Understand the purpose of the following Ada keywords:
   a. **with**
   b. **use**
   c. **procedure**
   d. **is**
   e. **begin**
   f. **end**
6. Gain an understanding of the conventional techniques used to make Ada code more readable and understandable.
7. Gain an understanding of the structure of an Ada program.

V. **LEARNING ACTIVITIES:**

1. Take notes on lecture presented by Instructor.
2. Participate in class discussion of presented lecture.
3. CAI Assignment - Block II
   AETECH *"Ada Training Environment"* or *"IntegrAda"* with *"On-Line Training and Reference Module"*.

   Read & take notes on the following sections:

   Block II, Lesson 1, Topic 1
   a. The Basic Ada Program.

VI. **SPECIAL RESOURCES:**

AETECH *"Ada Training Environment"* and *"IntegrAda"* with *"On-Line Training and Reference Module"*.

Skansholm, Ada From The Beginning, Addison - Wesley, 1988, pg. 30-31.

VII. **PRESENTATION**

A. Introduction

1. Put on board a flowchart of "Hello Program".

B. Instructional Topics and Key Points

TOPIC	KEY POINT
1. "Hello Program"	
`with Text_IO;`	1a. Imports to main procedure "Hello" Text_IO resources.
`use Text_IO`	1b. Uses abbreviated notation in lieu of extended dot notation. Note: Instructor should also show same program using extended dot notation.
`procedure Hello is`	1c. Name of main procedure.
`begin`	1d. Begin execution of program.
`put ("HELLO THERE");`	1e. Output to screen.
`end Hello;`	1f. End execution of program.
	SYNTAX NOTES
	a. note upper and lower case style and non-sensitivity.
	b. note ; delimiter.
	c. note () and "" for put (see Text_IO package).
2. Packages	2a. Definition - A collection of logically related program resources grouped together.
	2b. Use the predefined package Text_IO to demonstrate one such package as per 2a. above.
	2c. Identify context clause as the statement to be used to gain access to a package.
3. Comments	3a. Explain why comments are important.
	3b. examples of comments for "Hello Program".
	3c. Handout for program headers (HII.A.1).

TOPIC	KEY POINT
4. Keywords	4a. **with** - specifies the package to be made visible within another unit. 4b. **use** - Informs compiler program that programmer will not be using extended dot notation. 4c. **procedure** - One form of Ada subprogram; the other form is a function. A procedure specifies a sequence of actions, and is invoked by a procedure call statement. 4d. **is** - tells what items are given. 4e. **begin** - procedure execution starts here. 4f. **put** - procedure provided within Text_IO, puts a string to the screen. 4g. **end** - stop execution of main procedure.
5. Program Structure	5a. Show the structure for a typical Ada program. (Handout HII.A.2)

PROGRAM HEADERS

The following format should be incorporated into each of your programs to provide necessary documentation, and also provide identification information for you and your Instructor.

```
                  --*********************--;
                  --*   Program Name    *--;
                  --*********************--;

-- Author's Name    : __________;
-- Date             : ______________ ;
-- Assignment Number: ______________ ;

--------------    Program Executive     -----------------;

-- Provide a brief but accurate description of what the  ;
-- program does, and any other information which may be  ;
-- useful in describing your program.                    ;
---------------------------------------------------------;
```

HII.A2

A TYPICAL STRUCTURE FOR AN ADA MAIN PROCEDURE

```
   (Context Clauses)
with _____;   use _____ ;

procedure NAME is

   (Place declarations here);

begin


   (Program code);

end NAME;
```

LABORATORY EXPERIMENT

I. **BLOCK:** II - "Fundamentals of Ada Programming"

II. **UNIT:** A

III. **LAB NUMBER:** 05

IV. **LAB TITLE:** "A Basic Ada Program"

V. **STUDENT OBJECTIVES:** At the completion of this experiment, the student should be able to:

1. Write a simple Ada program.
2. Compile, bind, debug, and execute a simple Ada program.
3. Gain an understanding of what occurs in step (2) above.

VI. **REQUIRED MATERIALS:**

1. Note taking materials.
2. AETECH *"IntegrAda"* with *"On-Line Training and Reference Module"*.
3. Student Data Disk.

VII. **PROCEDURE**

1. Using the editor environment, type in the following code and save it to a file called LAB5.ADA

   Note: Be sure to include information from handout HII.A here!

```
with Text_IO; use Text_IO;
procedure <procedure_name> is
begin
  put("HELLO!");
  end <procedure_name>;
```

2. Compile, debug, bind, and execute the program.
3. Make a print out of your program and executable code to turn in to your Instructor.
4. Power down computer, and clean up area.

```
          --*****************************--;
          --*    A Basic Ada Program   *--;
          --*****************************--;

        -- Author's Name       : TEACHER GUIDE ;
        -- Assignment Number   : LAB # II.A ;

-------------     Program Executive     ----------------;
 -- Below is a solution for Lab # II.A. This solution may
 -- be used by the instructor as a guide for helping
 -- students complete the laboratory assignment.
--------------------------------------------------------;


    with Text_IO;  use Text_IO;

    procedure WELCOME is
    begin
      put( "HELLO!" );
    end WELCOME;
```

INFORMATION LESSON PLAN

I. **BLOCK:** II - "Fundamentals of Ada Programming"

II. **UNIT:** B

III. **LESSON TITLE:** "Existing Packages"

IV. **LESSON OBJECTIVES:** At the completion of this lesson, the student should be able to:

1. Gain an understanding of how to use existing code from Ada packages in new Ada programs.
2. Gain an understanding of the following keywords:

    a. **with**
    b. **use**

3. Utilize simple subprograms from existing packages to perform fundamental screen and keyboard operations needed for users to view and enter data to Ada programs.

V. **LEARNING ACTIVITIES:**

1. Take notes on lecture presented by Instructor.
2. Participate in class discussion of presented lecture.
3. CAI Assignment - Block II
   AETECH *"Ada Training Environment"* or *"IntegrAda"* with *"On-Line Training and Reference Module"*.

   Read & take notes on the following sections:

   Block II, Lesson 1, Topic 6

   a. Using existing packages.

VI. **SPECIAL RESOURCES:**
AETECH *"Ada Training Environment"* and *"IntegrAda"* with *"On-Line Training and Reference Module"*.

Johnson, The Ada Primer, McGraw-Hill, 1985, pg. 61

VII. PRESENTATION

A. Introduction

1. Handout copies of existing package specifications for SCREEN, KEYBOARD, and COLORS, and explain to students all this programming has already been done for the student.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
1. Existing Packages	1a. TEXT_IO - Standard Ada package for input and output of characters and strings of characters. Does not include cursor, screen, color, function keys, or simple keypresses. Used mainly for file operations.
	1b. SCREEN - Existing package used to handle simple cursor and screen operations.
	1c. COLOR - Existing package used to set foreground and background colors for other operations found in package SCREEN above.
	1d. KEYBOARD - Existing package used to get and identify keys pressed by the user.
2. Keywords	2a. **with** - Makes an existing package visible to your program (said to import an existing package to a main procedure).
	2b. **use** - Tells compiler that the programmer will not be using extended dot notation.
3. Utilizing Subprograms	3a. Give example using resources of several existing packages together to clear the screen in a color, set the cursor, print a message, and get a response from the user.

LABORATORY EXPERIMENT

I. **BLOCK:** II - "Fundamentals of Ada Programming"

II. **UNIT:** B

III. **LAB NUMBER:** 06

IV. **LAB TITLE:** "Existing Packages"

V. **STUDENT OBJECTIVES:** At the completion of this experiment, the student should be able to:

1. Identify the existing packages which the following program utilizes.
2. Use existing packages for simple input and output of data.

VI. **REQUIRED MATERIALS:**

1. Note taking materials.
2. AETECH *"IntegrAda"* with *"On-Line Training and Reference Module"*.
3. Student Data Disk.
4. Specifications for packages SCREEN, KEYBOARD, COLORS, TEXT_IO.

VII. **PROCEDURE**

1. Given the following simple program using typical input and output, identify the source package from which each of the following bold faced procedures and data structures come, by using extended dot notation. That is, if "SET_CURSOR" is found in package SCREEN, then rewrite the procedure to read "SCREEN.SET_CURSOR".
2. Given the specification for Ada package TEXT_IO, list those subprograms which are also available without other existing packages.

(Program on next page).

Save your program as LAB6.ADA.

```
with SCREEN,KEYBOARD,COLORS,TEXT_IO;
use SCREEN, KEYBOARD, COLORS;
procedure TRY_IT is
   KEY:A_KEY;
   CH:CHARACTER;
begin
   SET_BACKGROUND(BLUE);
   SET_FOREGROUND(YELLOW);
   CLEAR_SCREEN;
   loop
      SET_CURSOR(25,1);
      PUT("ENTER Any Key to Continue or <ESC> to
      Escape=>");
      PRESS(A_KEY,CH);
      PUT(A_KEY'IMAGE(KEY));
      exit when KEY=ESC;
   end loop;
end TRY_IT;
```

3. Compile, debug, bind, and execute the program.
4. Print out a copy of your program, and your executable output to turn in to your Instructor.
5. Power down computer, and clean up area.

```
          --****************************--;
          --*    Existing Packages     *--;
          --****************************--;

-- Author's Name            : TEACHER GUIDE ;
-- Assignment Number        : LAB # II.B ;

------------------   Program Executive    ----------------
-- Below is a solution for Lab # II.B. This solution may be
-- used by the instructor as a guide for helping students
-- complete the laboratory assignment
---------------------------------------------------------

with SCREEN, KEYBOARD, COLORS, TEXT_IO;
use  SCREEN, KEYBOARD, COLORS;

procedure TRY_IT is
 KEY: KEYBOARD.A_KEY;
 CH : CHARACTER;
begin
 COLORS.SET_BACKGROUND( BLUE );
 COLORS.SET_FOREGROUND( YELLOW );
 SCREEN.CLEAR_SCREEN;
 Loop
    SCREEN.SET_CURSOR( 25, 1 );
   SCREEN.PUT("Any Key to Continue or <ESC> to Escape =>");
    KEYBOARD.PRESS( KEY, CH );
    SCREEN.PUT( A_KEY'IMAGE( KEY ) );
    exit when KEY = ESC;
 end Loop;
end TRY_IT;
```

```
with SCREEN, KEYBOARD, COLORS, TEXT_IO;
use SCREEN, KEYBOARD,COLORS;
procedure TRY_IT is
 KEY: KEYBOARD.A_KEY;
 CH : CHARACTER;
begin
 SET_BACKGROUND( BLUE );
 SET_FOREGROUND( YELLOW );
 CLEAR_SCREEN;
 Loop
      SET_CURSOR( 25, 1 );
  TEXT_IO.PUT("Any Key to Continue or <ESC> to Escape =>");
    PRESS( KEY, CH );
    TEXT_IO.PUT( A_KEY'IMAGE( KEY ) );
    exit when KEY = ESC;
 end Loop;
end TRY_IT;
```

INFORMATION LESSON PLAN

I. **BLOCK:** II - "Fundamentals of Ada Programming"

II. **UNIT:** C

III. **LESSON TITLE:** "Package Text_IO"

IV. **LESSON OBJECTIVES:** At the completion of this lesson, the student should be able to:

1. Use the non-generic resources provided within Text_IO.
2. Instantiate and use the generic packages Integer_IO, Float_IO, Fixed_IO, and Enumeration_IO.
3. Describe and use the width and base parameters provided for Integer_IO.
4. Describe and use the Fore, Aft, and Exp parameters provided for Fixed_IO and Float_IO.

V. **LEARNING ACTIVITIES:**

1. Take notes on lecture presented by Instructor.
2. Participate in class discussion of presented lecture.
3. CAI Assignment - Block VI
   AETECH *"Ada Training Environment"* or *"IntegrAda"* with *"On-Line Training and Reference Module"*.

   Read & take notes on the following sections:

   Block VI, Lesson 4, Topics 1-11

   a. Package Text_IO.
   b. Instantiation.
   c. Console input/output.
   d. File handling.
   e. Characters and new_line.
   f. Strings.
   g. Working with strings.
   h. Integer_IO.
   i. Float_IO.
   j. Fixed_IO.
   k. Enumeration_IO.

VI. SPECIAL RESOURCES:

AETECH *"Ada Training Environment"* and *"IntegrAda"* with *"On-Line Training and Reference Module"*.

VII. PRESENTATION

A. Introduction

1. Inform students that package Text_IO provides the basic resources for input/output of text or data text files. The standard default input file is the keyboard. The standard default output file is the screen.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
1. Text_IO	1a. Is a predefined package which contains subprogram resources (for I/O of strings and characters) and generic package for I/O of Integers, Floats, Fixed, and Enumerated type objects.
2. Generics	2a. Integer_IO, Fixed_IO, Float_IO, and Enumeration_IO are the names of generic packages within Text_IO which must be instantiated in order to gain access to the I/O resources for their respective types.
3. Integer_IO	3a. Width - defaults actual width of the type when the package was instantiated. 3b. Base - an optional parameter which allows for working with different base number systems. Default is 10. (where subtype Number_Base is Integer Range 2..16)

B. Instructional Topics and Key Points

TOPIC	KEY POINT
4. Fixed_IO and Float_IO	4a. Fore - An optional parameter which specifies the number of character positions to the left of the decimal. For Floating Point types, Fore defaults to 2; for fixed, number in type given. 4b. Aft - An optional parameter which specifies the number of character positions to the right of the decimal. For floating point types, defaults to number in type -1; for fixed, number in type. 4c. Exp - An optional parameter which specifies the number of character positions to use for the exponent part.

LABORATORY EXPERIMENT

I. **BLOCK:** II - "Fundamentals of Ada Programming"

II. **UNIT:** C

III. **LAB NUMBER:** 07

IV. **LAB TITLE:** "Ohm's Law"

V. **STUDENT OBJECTIVES:** At the completion of this experiment, the student should be able to:

1. Compile, bind, debug, and execute an Ada program which calculates voltage based on user input values for current and resistance. Ohm's Law states that voltage (volts) is equal to current (amps) multiplied by resistance (ohm's).

   $$V = I * R$$

2. Modify the program so that current is calculated using floating point types.
3. Modify the program so that resistance is calculated using floating point types.

VI. **REQUIRED MATERIALS:**

1. Note taking materials.
2. AETECH *"IntegrAda"* with *"On-Line Training and Reference Module"*.
3. Student Data Disk.

VII. **PROCEDURE**

1. Using the editor environment, type in the following code and save it to a file called LAB7A.ADA. Be sure that you include all information as per handout HII.A. "Program Headers".

```
             --*************************************--;
             --*  Ohm's Law, Voltage Calculation   *--;
             --*************************************--;

    -- Author's Name          :
    -- Assignment Number      : LAB # II.C ;

----------------    Program Executive    ----------------


---------------------------------------------------------
   with TEXT_IO;  use TEXT_IO;

   procedure OHMS is

     package IntegerIO is new INTEGER_IO( INTEGER );

     V, I, R: INTEGER;
   begin
     PUT( "Enter Current (in Amps):  ");
     IntegerIO.GET( I );
     NEW_LINE;
     PUT( "Enter Resistance (in Ohms):  ");
     IntegerIO.GET( R );
     NEW_LINE;
     V := I * R;
     NEW_LINE;   NEW_LINE;
     PUT( "***** Voltage (in Volts) = " );
     IntegerIO.PUT( V, Width => 1 );
     PUT( " *****");
     NEW_LINE;
  end OHMS;
```

2. Compile, debug, bind, and execute the program.

3. Print out a copy of your program, and your executable output to turn in to your Instructor.

4. Modify the program to calculate current based on input values of voltage and resistance. Change the type of current, voltage, and resistance to float. Be sure that you instantiate the required generic package within Text_IO. Follow steps 2-3 above, saving this new program as LAB7B.ADA.

5. Modify the program in step 4 to calculate resistance based on input values of current and voltage. Follow steps 2-3 above, saving this new program as LAB7C.ADA.

6. Power down computer, and clean up area.

```
--***************************************--;
--*  Ohm's Law, Current Calculation   *--;
--***************************************--;

-- Author's Name           : TEACHER GUIDE ;
-- Assignment Number       : LAB # II.C, Procedure 4

------------------   Program Executive   ----------------
-- Below is a solution for Lab # II.C, procedure number 4.
-- This solution may be used by the instructor as a guide
-- for helping students complete the laboratory assignment.
---------------------------------------------------------

with TEXT_IO;  use TEXT_IO;

procedure OHMS2 is

   package FloatIO is new FLOAT_IO( FLOAT );
   use FloatIO;

   V, I, R: FLOAT;

begin
   PUT( "Enter Voltage (in Volts):  ");
   GET( V );
   NEW_LINE;
   PUT( "Enter Resistance (in Ohms):  ");
   GET( R );
   NEW_LINE;
   I := V / R;
   NEW_LINE;   NEW_LINE;
   PUT( "***** Current (in Amps) = " );
   PUT( I, Aft => 2, Exp => 0 );
   PUT( " *****");
   NEW_LINE;
end OHMS2;
```

```
      --****************************************--;
      --*   Ohm's Law, Resistance Calculation  *--;
      --****************************************--;

-- Author's Name          : TEACHER GUIDE ;
-- Assignment Number      : LAB # II.C, Procedure 5;

-----------------    Program Executive    ---------------
-- Below is a solution for Lab # II.C, procedure 5.
-- This solution may be used by the instructor as a guide
-- for helping students complete the laboratory
assignment.
---------------------------------------------------------
with TEXT_IO;  use TEXT_IO;

procedure OHMS3 is

   package FloatIO is new FLOAT_IO( FLOAT );
   use FloatIO;

   V, I, R: FLOAT;

begin
   PUT( "Enter Current (in Amps):  ");
   GET( I );
   NEW_LINE;
   PUT( "Enter Voltage (in Volts):  ");
   GET( V );
   NEW_LINE;
   R := V / I;
   NEW_LINE;   NEW_LINE;
   PUT( "***** Resistance (in Ohms) = " );
   PUT( R, Aft => 2, Exp => 0 );
   PUT( " *****");
   NEW_LINE;
end OHMS3;
```

INFORMATION LESSON PLAN

I. **BLOCK:** II "Fundamentals of Ada Programming"

II. **UNIT:** D

III. **LESSON TITLE:** "Package Standard"

IV. **LESSON OBJECTIVES:** At the completion of this lesson, the student should be able to:

1. Define the type Boolean.
2. Understand the following functions, and be able to diagram the truth tables for them:

   not
   or
   and
   xor

3. Identify the operations that are available for mixed types, and describe the returned results.
4. Define the following predefined subtypes:

   Natural
   Short_Integer
   Short_Float
   Positive
   Long_Integer
   Long_Float

5. Identify what predefined operations are provided by package STANDARD for strings, and identify what is returned by these operations.

V. **LEARNING ACTIVITIES:**

1. Take notes on lecture presented by Instructor.
2. Participate in class discussion of presented lecture.

V. **LEARNING ACTIVITIES** (continued):

3. CAI Assignment - Block VI
AETECH *"Ada Training Environment"* or *"IntegrAda"* with *"On-Line Training and Reference Module"*.

Read & take notes on the following sections:

Block VI, Lesson 1, Topics 1-10

a. Using package STANDARD.
b. Boolean functions.
c. Integer functions.
d. Float functions.
e. Mixed functions.
f. Type Character.
g. ASCII control constants.
h. ASCII character constants.
i. Predefined subtypes.
j. String functions.

VI. SPECIAL RESOURCES:

AETECH *"Ada Training Environment"* and *"IntegrAda"* with *"On-Line Training and Reference Module"*.

VII. PRESENTATION

A. Introduction

1. Define package STANDARD as a package which provides many primary operators for the predefined Ada types (i.e. '+' for Integers). Remind students that they don't have to instantiate the package because it is not a generic. Package STANDARD is automatically "withed" and "used" by the compiler for all units.

B. Instructional Topics and Key Points

<table>
<tr><th>TOPIC</th><th>KEY POINT</th></tr>
<tr><td>1. Type BOOLEAN</td><td>1a. A predefined type which can have a value of either true or false. Has the operators =, >, <, etc. defined for it. (give examples).</td></tr>
<tr><td>2. Logical Operators and their Truth Values</td><td>2a. not (for X) - the value of X is reversed.
2b. and (X AND Y) -both X and Y must be true to return true:
<table>
<tr><th>X</th><th>Y</th><th>Result</th></tr>
<tr><td>F</td><td>F</td><td>F</td></tr>
<tr><td>F</td><td>T</td><td>F</td></tr>
<tr><td>T</td><td>F</td><td>F</td></tr>
<tr><td>T</td><td>T</td><td>T</td></tr>
</table>
</td></tr>
</table>

B. Instructional Topics and Key Points

<table>
<tr><th>TOPIC</th><th>KEY POINT</th></tr>
<tr><td>2. Logical Operators and their Truth Values (Continued)</td><td>2c. or (X OR Y) - either X or Y must be true to return true:
<table>
<tr><th>X</th><th>Y</th><th>Result</th></tr>
<tr><td>F</td><td>F</td><td>F</td></tr>
<tr><td>F</td><td>T</td><td>T</td></tr>
<tr><td>T</td><td>F</td><td>T</td></tr>
<tr><td>T</td><td>T</td><td>T</td></tr>
</table>
2d. xor (X OR Y but not both) - either x or y can be true to return true, but not both of them:
<table>
<tr><th>X</th><th>Y</th><th>Result</th></tr>
<tr><td>F</td><td>F</td><td>F</td></tr>
<tr><td>T</td><td>F</td><td>T</td></tr>
<tr><td>F</td><td>T</td><td>T</td></tr>
<tr><td>T</td><td>T</td><td>F</td></tr>
</table>
</td></tr>
<tr><td>3. Mixing Types</td><td>3a. The following operations are available for mixing real numbers with integers: Multiplication - defined for either real or integer as first number. Returns real. Division - denominator is integer. Returns real.</td></tr>
<tr><td>4. Predefined Subtypes</td><td>4a. **Natural** 0..integer'last
Positive 1..integer'last
Short_Integer, Long_Integer
Short_Float, Long_Float - all are implementation defined, where computer defines boundary based on its own internal structure.</td></tr>
<tr><td>5. Strings</td><td>5a. Definition - unconstrained array of characters.
5b. Operations:
=, <, >, <=, >=, /=
(give examples)
5c. & - will concatenate any combination of strings and characters. Returns string.</td></tr>
</table>

B. Instructional Topics and Key Points

TOPIC	KEY POINT
6. Arrays	6a. Basic operations include assignment, membership tests, indexed components, qualification, and explicit conversion. For one dimensional arrays, slices and string operations are supported.

LABORATORY EXPERIMENT

I. **BLOCK:** II - "Fundamentals of Ada Programming"

II. **UNIT:** D

III. **LAB NUMBER:** 08

IV. **LAB TITLE:** "Working with Package Standard"

V. **STUDENT OBJECTIVES:** At the completion of this experiment, the student should be able to:

1. Compile, bind, debug, and execute an Ada Program which outputs Natural and Positive type objects.

VI. **REQUIRED MATERIALS:**

1. Note taking materials.
2. AETECH *"IntegrAda"* with *"On-Line Training and Reference Module"*.
3. Student Data Disk.

VII. **PROCEDURE**

1. Write a procedure which declares Object_1 as a Positive Integer, and Object_2 as a Natural Integer. Output the smallest value possible ('First) for each of the objects. Save this program as LAB8.ADA.
2. Compile, debug, bind, and execute the program.
3. Print out a copy of your program, and your executable output to turn in to your Instructor.
4. Power down computer, and clean up area.

```
        --*****************************************--;
        --*    Working with Package Standard      *--;
        --*****************************************--;

   -- Author's Name          : TEACHER GUIDE ;
   -- Assignment Number      : LAB # II.D ;

--------------------   Program Executive   -------------
 -- Below is a solution for Lab # II.D.  This solution may
 -- be used by the instructor as a guide for helping
 -- students complete the laboratory assignment.
--------------------------------------------------------
   with TEXT_IO;   use TEXT_IO;

   procedure PRINT_FIRSTS is

      package PositiveIO is new INTEGER_IO( POSITIVE );
      package NaturalIO is new INTEGER_IO( NATURAL );

      Object_1 : POSITIVE;
      Object_2 : NATURAL;
   begin
      Object_1 := POSITIVE'FIRST;
      Object_2 := NATURAL'FIRST;
      NEW_LINE;
      PUT( "Smallest Possible POSITIVE value is: " );
      PositiveIO.PUT( Object_1, Width => 2 );
      NEW_LINE; NEW_LINE;
      PUT( "Smallest Possible NATURAL value is:  " );
      NaturalIO.PUT( Object_2, Width => 2 );
      NEW_LINE;
   end PRINT_FIRSTS;
```

INFORMATION LESSON PLAN

I. **BLOCK:** II - "Fundamentals of Ada Programming"

II. **UNIT:** E

III. **LESSON TITLE:** "Simple Declarations"

IV. **LESSON OBJECTIVES:** At the completion of this lesson, the student should be able to:

1. Define variable objects.
2. Define the following types:
    - a. Integer (including positive and natural).
    - b. Real (including fixed and float).
    - c. Character.
    - d. String.
3. Perform simple variable object declarations with initializations.
4. Perform simple variable object assignment.

V. **LEARNING ACTIVITIES:**

1. Take notes on lecture presented by Instructor.
2. Participate in class discussion of presented lecture.
3. CAI Assignment - Block II
   AETECH *"Ada Training Environment"* or *"IntegrAda"* with *"On-Line Training and Reference Module"*.

   Read & take notes on the following sections:

   Block II, Lesson 1, Topic 2

   a. Simple Declarations.

VI. **SPECIAL RESOURCES:**

AETECH *"Ada Training Environment"* and *"IntegrAda"* with *"On-Line Training and Reference Module"*.

VII. PRESENTATION

A. Introduction

1. Use an example of a person's age being a variable; include:

String - July 1, 1960
Integer - 711960
Real - 711960.6

B. Instructional Topics and Key Points

TOPIC	KEY POINT
1. Variables	1a. In Ada, variables are one type of objects.
	1b. Variables provide a way to save and retrieve data.
	1c. Variables of different types cannot be mixed implicitly.
2. Variable Types	2a. Discrete: Integer - A signed (+/-) whole #.(no fractions or decimals) may utilize isolated embedded underscores. Two predefined subtypes: Natural-includes 0 Positive-doesn't include 0.
	2b. Real - A signed (+/-) approximation of a number with a fractional or decimal part. Two types: Fixed - Real numbers where approximation's error bound is specified as an absolute value. Error bound is called the delta of the fixed point type. Float - Real numbers where approximation's error bound is specified by a minimum number of significant decimal digits.
	2c. Character -A digit, letter or some other single symbol.
	2d. String - One or more characters; a one dimensional unconstrained array whose components are characters.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
3. Variable Object Declaration	3a. Select a meaningful identifier to reference the variable. 3b. Specify the type of object that the variable may contain. (give examples) 3c. Optionally assign the object an initial value. Identifiers must start with alpha character, may be any reasonable length, may contain letters/digits/& underscores, no blanks, no 2 adjacent underscores (single embedded underscores; no trailing underscores). 3d. Variables have no value unless initialized, or given a value in an assignment statement.
4. Variable Assignment	4a. Use := to make assignment of right side, to variable object on left side. 4b. Identifier appears on left side of assignment statement. 4c. Expression on right side must be of the same type as variable on left, because Ada is a strongly typed language. 4d. May make assignment in declaration. This is called initialization. (give examples).

LABORATORY EXPERIMENT

I. **BLOCK:** II - "Fundamentals of Ada Programming"

II. **UNIT:** E

III. **LAB NUMBER:** 09

IV. **LAB TITLE:** "Simple Declarations Worksheet"

V. **STUDENT OBJECTIVES:** At the completion of this experiment, the student should be able to:

1. Complete the worksheet "Simple Declarations".

VI. **REQUIRED MATERIALS:**

1. Writing Utensil.

VII. **PROCEDURE**

1. Complete the attached worksheet "Simple Declarations".

"Simple Declarations"

WORKSHEET

A. Perform the following operations:

1. Declare a variable of type integer.
2. Declare a variable of type string.
3. Declare a variable of type character.
4. Declare a variable of type natural.
5. Declare a variable of type positive.
6. Declare a variable of type fixed.
7. Declare a variable of type float.
8. Declare a variable of type string with 10 characters.
9. Declare a variable of type integer, and assign it an initial value of 10.
10. Declare a variable of type fixed, and assign it a value.

B. Identify whether the following operations and assignments are legal, or whether an error would occur. Circle L for legal and E for error. If E, explain why an error would result.

```
A:INTEGER;                 E:FIXED;
B:CHARACTER;               F:POSITIVE;
C:STRING(1..6);            G:NATURAL;
D:FLOAT;
```

```
 1. A:=10;          L        E
 2. A:=2041.2;      L        E
 3. C:=A+B;         L        E
 4. B:="A";         L        E
 5. D:=6.14;        L        E
 6. E=3.45          L        E
 7. B:='C';         L        E
 8. F:=-6.0;        L        E
 9. F:=0;           L        E
10. G:=0;           L        E
```

"Simple Declarations"

ANSWERS TO QUESTIONS

A. Perform the following operations:

For part A, 1-10, each individual student will provide a different answer, therefore teacher should follow variable declaration guidelines in determining the correctness of the provided student answer.

B. Identify whether the following operations and assignments are legal, or whether an error would occur. circle L for legal and E for error. If E, explain why an error would result.

```
A:INTEGER;                E:FIXED;
B:CHARACTER;              F:POSITIVE;
C:STRING(1..6);           G:NATURAL;
D:FLOAT;
```

1. A:=10; [✓ L] E
2. A:=2041.2; L [✓ E] - number cannot have decimal point if it is declared as integer.
3. C:=A+B; L [✓ E] - cannot perform operations on objects of different type.
4. B:="A"; L [✓ E] - quotes are around strings, not characters; apostrophes are around characters.
5. D:=6.14; [✓ L] E
6. E=3.45 L [✓ E] - missing colon, missing semicolon.
7. B:='C'; [✓ L] E
8. F:=6.0; L [✓ E] - number not positive
9. F:=0; L [✓ E] - positive doesn't include 0.
10. G:=0; [✓ L] E

INFORMATION LESSON PLAN

I. **BLOCK:** II - "Fundamentals of Ada Programming"

II. **UNIT:** F

III. **LESSON TITLE:** "Using Existing Packages; Parameters, Specifications, and Calls"

IV. **LESSON OBJECTIVES:** At the completion of this lesson, the student should be able to:

1. Identify and understand a specification for an existing package.
2. Identify formal and actual parameters of subprogram and function specifications.
3. Identify the three modes of parameter passing and understand how to use each mode of parameter passing.
4. Identify "named" and "positional" notation for use with calling subprograms.

V. **LEARNING ACTIVITIES:**

1. Take notes on lecture presented by Instructor.
2. Participate in class discussion of presented lecture.
3. CAI Assignment - Block II
   AETECH *"Ada Training Environment"* or *"IntegrAda"* with *"On-Line Training and Reference Module"*.

   Read & take notes on the following sections:

   Block II, Lesson 1, Topics 3-5

   a. Parameters.
   b. Specifications.
   c. Calling procedures.

VI. **SPECIAL RESOURCES:**

AETECH *"Ada Training Environment"* and *"IntegrAda"* with *"On-Line Training and Reference Module"*.

Skansholm, Ada From The Beginning, Addison-Wesley, 1988.

VII. PRESENTATION

A. Introduction

1. Hand out specifications for any existing package, and have students look at those specifications {those things in () } and ask if anyone knows what it is.
   (Answer: Formal parameters and their modes and type).

B. Instructional Topics and Key Points

TOPIC	KEY POINT
1. Parameter	1a. Definition - The object (variable) used in passing values when a procedure or function is called. Values are passed from main or calling unit to or from the subprogram. There are two types of parameter lists. FORMAL - Those parameters listed in the specification of a function or subprogram, which will be used as holders for the data passed to/from the actual parameters. ACTUAL - Those parameters listed in the calling program, which will be used to pass data to/from the called function or subprogram's formal procedures, on a 1:1 basis.
2. Specification	2a. Definition - The portion of a package, procedure, or function which tells the user how to use, or interface with it. 2b. Lists parameters that are used by that package, function, or procedure. For a subprogram it lists the names, modes, and types of the parameters.
3. Modes of Parameters (As used with a main program as caller, and a procedure as the called unit).	3a. IN -A main procedure is said to "drive" system. When the main procedure calls the sub-program and the subprogram is executed, the actual parameter in the call gives value to the formal parameter. In the subprogram, the formal para-

B. Instructional Topics and Key Points

TOPIC	KEY POINT
3. Modes of Parameters (As used with a main program as caller, and a procedure as the called unit).	meter acts like a constant, therefore its value cannot be changed (acts like a literal in the called unit).
	3b. OUT - When the main program calls the subprogram and the subprogram is executed, the formal parameter's value is undefined (the formal parameter is a variable object in the main procedure). In the called unit, this formal, since it is undefined, may not appear on the right hand side of the assignment statement. The value of formal parameter, when assigned, will be passed back to actual the parameter upon completion of subprogram or function (sends value back to the main procedure).
	3c. IN OUT - The formal parameter has a value at the time of call. In a procedure, the formal parameter is used as an ordinary variable, whose value can be used & changed. If formal is changed, then actual will be changed when formal exports value upon completion of the subprogram.
4. Types of Association	4a. When a call is made the actual parameters are associated with formal parameters by either named association or positional association. NAMED ASSOCIATION - In named association, the name of the formal parameter is given in the actual call list followed by a "=>" sign followed by a value or variable. POSITIONAL ASSOCIATION - In positional association, the value of the actual parameter is passed via its position in relation to the formal specification list, on a 1:1 basis.

LABORATORY EXPERIMENT

I. **BLOCK:** II - "Fundamentals of Ada Programming"

II. **UNIT:** F

III. **LAB NUMBER:** 10

IV. **LAB TITLE:** "Combining Existing Packages"

V. **STUDENT OBJECTIVES:** At the completion of this experiment, the student should be able to:

1. Write a simple procedure which uses both the predefined Ada packages STANDARD and CALENDAR for computations, and other existing Ada SCREEN and KEYBOARD packages for input and output.

VI. **REQUIRED MATERIALS:**

1. Note taking materials.
2. AETECH *"IntegrAda"* with *"On-Line Training and Reference Module"*.
3. Student Data Disk.
4. AETECH IntegrAda or Alsys "AdaUser" Libraries.

VII. **PROCEDURE**

1. Using the provided example, enter the simple procedure which calculates the time difference between two user input responses, the existing packages SCREEN and KEYBOARD for user control and the instantiated package from TEXT_IO for output of the type DURATION.

Example:

```
with TEXT_IO,KEYBOARD,CALENDAR,SCREEN;
use KEYBOARD,CALENDAR,SCREEN;
procedure TIME_IT is
  START_TIME,FINISH_TIME:TIME;
  package DURATION_IO is new
  TEXT_IO.FIXED_IO(DURATION);
  use DURATION_IO;
  KEY:KEYBOARD.A_KEY;
  CH:CHARACTER;
begin
  loop
    CLEAR_SCREEN;
    PUT_LINE("Press any key to start timing, or
    <ESC> to Quit=>");
    PRESS(KEY,CH);
    START_TIME:=CLOCK;
    exit when KEY=ESC;
    PUT_LINE("Press any key to stop
    timing =====>");
    PRESS(KEY,CH);
    FINISH_TIME:=CLOCK;
    PUT("The time elapsed between start and
    stop was =====>");
    DURATION_IO.PUT(FINISH_TIME-START_TIME);
    delay 3.0;
  end loop;
end TIME_IT;
```

Save your program as LAB10.ADA.

2. Compile, debug, bind, and execute the program.

3. Print out a copy of your program, and your executable output to turn in to your Instructor.

4. Power down computer, and clean up area.

INFORMATION LESSON PLAN

I. **BLOCK:** II - "Fundamentals of Ada Programming"

II. **UNIT:** G

III. **LESSON TITLE:** "Data Types"

IV. **LESSON OBJECTIVES:** At the completion of this lesson, the student should be able to:

1. Identify two primary declaration statements of the Ada programming language (Object declarations and Type declarations).
2. Declare objects and types and understand the operations that may be performed on them.
3. Identify the operation limitations for objects and types.
4. Understand the use of a declare statement, and why and when declarations are performed within a program.

V. **LEARNING ACTIVITIES:**

1. Take notes on lecture presented by Instructor.
2. Participate in class discussion of presented lecture.
3. CAI Assignment - Block III
   AETECH *"Ada Training Environment"* or *"IntegrAda"* with *"On-Line Training and Reference Module"*.

   Read & take notes on the following sections:
   Block III, Lesson 1, Topics 1-5

   a. Data structures.
   b. Type declarations.
   c. Operations on types.
   d. Limitations on operations.
   e. Location of declarations.

VI. **SPECIAL RESOURCES:**

AETECH *"Ada Training Environment"* and *"IntegrAda"* with *"On-Line Training and Reference Module"*.

VII. PRESENTATION

A. Introduction

1. Compare objects and types to packing boxes in that an object is the name of the empty box, and the type is what the box may hold. The packing of boxes is done using initialization or assignment.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
1. Primary Declarations	1a. Two primary declarations: a. types b. objects 1b. Types - defines a set of operations and values that an object may have. 1c. Object - (variable) - entity which can take on a value and use the operations defined for its type.
2. Declaring objects and types	2a. Ensure that the student can do this by showing many examples.
3. Limitations	3a. Objects cannot be assigned values of other types; no mixing of types (apples and oranges). This is Ada's strong typing characteristic. 3b. Objects cannot have operations performed on them that are not specified for that type (refer to package STANDARD for operations).
4. Location of Declarations	4a. Declarations must come before any attempt to use them. 4b. Declarations can occur in specification part of package, or declaration part of block, package body, or subprogram.
5. Declare Statement	5a. May use declare statement to declare objects later in structure. (i.e. inside a local block). 5b. Three Components to Declare Statement: "Declare" followed by declarations then "begin" and "end", where begin/end represent the local block.

INFORMATION LESSON PLAN

I. **BLOCK:** II - "Fundamentals of Ada Programming"

II. **UNIT:** H

III. **LESSON TITLE:** "Ada Scalar Types"

IV. **LESSON OBJECTIVES:** At the completion of this lesson, the student should be able to:

1. Identify the predefined Ada types including Integer, Float, Fixed, Character, and Boolean.
2. Understand the use of explicit typing, and be able to declare explicit types.
3. Define Integer, Float, and Fixed types.
4. Be able to declare objects of predefined data types.

V. **LEARNING ACTIVITIES:**

1. Take notes on lecture presented by Instructor.
2. Participate in class discussion of presented lecture.
3. CAI Assignment - Block III
   AETECH *"Ada Training Environment"* or *"IntegrAda"* with *"On-Line Training and Reference Module"*.

   Read & take notes on the following sections:

   Block III, Lesson 2, Topics 1-5, 7, 8.

   a. Predefined Integer types.
   b. Explicit typing.
   c. Assignments within explicit ranges.
   d. Floating point types.
   e. Fixed point types.
   f. Enumeration type Boolean.
   g. Enumeration type Character.

VI. **SPECIAL RESOURCES:**

AETECH *"Ada Training Environment"* and *"IntegrAda"* with *"On-Line Training and Reference Module"*.

VII. PRESENTATION

A. Introduction

1. Have students look at package STANDARD noting the various types which have been predefined by this package, and the operations which have been predefined for these included types.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
1. Predefined Discrete Types	1a. Integer, Character, Boolean. 1b. **Integer** - positive or negative whole #, machine implementation dependent. 1c. **Boolean** - enumeration type which can have two values: True or False. 1d. **Character** - A single alpha, digit, or other special symbol (the ASCII set). Enclosed by apostrophes (' ').
2. Predefined Real Types	2a. **Float** -approximation of a real number with a declared degree of decimal digits. 2b. **Fixed** - explicitly defined error bound or delta, used where accuracy is important.
3. Predefined Subtypes	3a. **Natural** - (0..'Last). 3b. **Positive** - (1..'Last).
4. Unconstrained Array Type	4a. **String** - an unconstrained array of characters. Enclosed in quotes. (" ").
5. Fixed Type	5a. **Duration.**
6. Explicit Typing	6a. Definition - make a certain type have more constraints than normal. 6b. Show why to use (accuracy, limits erroneous input, etc). 6c. Integer - range options. Float - digits & range options. Fixed - delta & range options.

LAB II.H

LABORATORY EXPERIMENT

I. **BLOCK:** II - "Fundamentals of Ada Programming"

II. **UNIT:** H

III. **LAB NUMBER:** 11

IV. **LAB TITLE:** "Scalar Types"

V. **STUDENT OBJECTIVES:** At the completion of this experiment, the student should be able to:

1. Write a procedure which declares and uses scalar type objects, and provides explicit type conversion.

VI. **REQUIRED MATERIALS:**

1. Note taking materials.
2. AETECH *"IntegrAda"* with *"On-Line Training and Reference Module"*.
3. Student Data Disk.

VII. **PROCEDURE**

1. Write a procedure which declares two objects. Object1 is of type Integer, and Object2 is of type Float. Provide Object2 with an initial value of 20.0. Prompt the user to enter an integer from 0 to 20 and calculate the percentage of the user input number to Object2 by:

   Percentage = (Object1/Object2) * 100

   Print to the screen the value of percentage.

   Save your program as LAB11.ADA.

2. Compile, debug, bind, and execute the program.
3. Print out a copy of your program and executable code to be turned in to your Instructor.
4. Power down computer, and clean up area.

```
                    --***********************--;
                    --*    Scalar Types     *--;
                    --***********************--;

  -- Author's Name          : TEACHER GUIDE ;
  -- Assignment Number      : LAB # II.H ;

-------------------   Program Executive   ---------------
-- Below is a solution for Lab # II.H. This solution may
-- be used by the instructor as a guide for helping
-- students complete the laboratory assignment.
--------------------------------------------------------
   with TEXT_IO;   use TEXT_IO;

   procedure Scalars is

      Object1 : INTEGER;
      Object2 : constant FLOAT := 20.0;
      Percentage : FLOAT;

      package FloatIO is new FLOAT_IO( FLOAT );
      package IntegerIO is new INTEGER_IO( INTEGER );

   begin
      put( "Please enter an integer value in the range 0 to
            20: " );
      IntegerIO.get( Object1 );
      NEW_LINE;  NEW_LINE;
      Percentage := ( FLOAT( Object1 ) / Object2 ) * 100.0;
      put( "Your input value is " );
      FloatIO.put( Percentage, Aft => 2, Exp => 0 );
      put( " percent of ");
      FloatIO.put( Object2, Aft => 2, Exp => 0 );
      put_line( "." );
   end Scalars;
```

INFORMATION LESSON PLAN

I. **BLOCK:** II - "Fundamentals of Ada Programming"

II. **UNIT:** I

III. **LESSON TITLE:** "Enumeration Types"

IV. **LESSON OBJECTIVES:** At the completion of this lesson, the student should be able to:

1. Define enumeration type.
2. Describe the ordering of enumeration types.
3. Declare enumerated types.
4. Use enumerated types.

V. **LEARNING ACTIVITIES:**

1. Take notes on lecture presented by Instructor.
2. Participate in class discussion of presented lecture.
3. CAI Assignment - Block III
   AETECH *"Ada Training Environment"* or *"IntegrAda"* with *"On-Line Training and Reference Module"*.

   Read & take notes on the following sections:

   Block III, Lesson 2, Topics 6

   a. User defined enumerated types.

VI. **SPECIAL RESOURCES:**

AETECH *"Ada Training Environment"* and *"IntegrAda"* with *"On-Line Training and Reference Module"*.

Skansholm, Ada From the Beginning, Addison- Wesley, 1988, pp. 172-173.

VII. **PRESENTATION**

A. Introduction

1. Show students how much easier it is to describe certain things using words instead of numbers (like the days of the week).

B. Instructional Topics and Key Points

TOPIC	KEY POINT
1. Enumeration Types	1a. Definition - allows for the meaningful description of real world entities.
	1b. Must follow rules for identifiers, may not be strings or numbers.
	1c. Use of enumerated types allow for better readability and understandability.
	1d. Ranges for enumerated types are declared, and a host of attributes are supported.
2. I/O of Enumerated Objects	2a. Must provide instantiation of generic package Enumerated_IO for I/O of enumerated_types, using the particular enumerated type as the actual parameter.

LABORATORY EXPERIMENT

I. **BLOCK:** II - "Fundamentals of Ada Programming"

II. **UNIT:** I

III. **LAB NUMBER:** 12

IV. **LAB TITLE:** "Enumeration Types"

V. **STUDENT OBJECTIVES:** At the completion of this experiment, the student should be able to:

1. Write a procedure which instantiates the package Enumeration_IO, and provides output for a user declared enumerated type.

VI. **REQUIRED MATERIALS:**

1. Note taking materials.
2. AETECH *"IntegrAda"* with *"On-Line Training and Reference Module"*.
3. Student Data Disk.

VII. **PROCEDURE**

1. Write a procedure which simulates a traffic light. Use the following declaration:

   ```
   type StopLightType is (Red, Yellow, Green);
   ```

   Start the light at green. Delay 5 seconds. Turn the light to yellow. Delay 3 seconds. Turn the light to red. Delay five seconds. Turn the light back to green. Save your lab as LAB12.ADA.

2. Compile, debug, bind, and execute the program.
3. Print out a copy of your program, and your executable output to turn in to your Instructor.
4. Power down computer, and clean up area.

```
             --****************************--;
             --*     Enumeration Types    *--;
             --****************************--;

   -- Author's Name           : TEACHER GUIDE ;
   -- Assignment Number       : LAB # II.I ;

------------------    Program Executive     ---------------
-- Below is a solution for Lab # II.I. This solution may be
-- used by the instructor as a guide for helping students
-- complete the laboratory assignment.
-----------------------------------------------------------
  with TEXT_IO;   use TEXT_IO;

  procedure Stop_Light is

    type StopLightType is ( Red, Yellow, Green );

    Signal : StopLightType;
    Message : STRING (1..22) := "The Traffic Light is: ";

   package LightIO is new ENUMERATION_IO( StopLightType );
   use LightIO;

begin
   put_line( "Hit <CTRL/C> to terminate Traffic Light." );
   loop
      Signal := Green;
      put( Message );
      put( Signal );
      NEW_LINE;
      delay( 5.0 );
      Signal := Yellow;
      put( Message );
      put( Signal );
      NEW_LINE;
      delay( 3.0 );
      Signal := Red;
      put( Message );
      put( Signal );
      NEW_LINE;
      delay( 5.0 );
   end loop;
end Stop_Light;
```

INFORMATION LESSON PLAN

I. **BLOCK:** II - "Fundamentals of Ada Programming"

II. **UNIT:** J

III. **LESSON TITLE:** "Derived Types"

IV. **LESSON OBJECTIVES:** At the completion of this lesson, the student should be able to:

1. Define derived type.
2. Identify the syntax associated with a derived type.
3. Identify the rules applicable for derived types.
4. Perform conversions between derived types and base types.

V. **LEARNING ACTIVITIES:**

1. Take notes on lecture presented by Instructor.
2. Participate in class discussion of presented lecture.
3. CAI Assignment - Block III
   AETECH *"Ada Training Environment"* or *"IntegrAda"* with *"On-Line Training and Reference Module"*.

   Read & take notes on the following sections:

   Block III, Lesson 5, Topics 1, 2.

   a. Creating derived types.
   b. Type conversion.

VI. **SPECIAL RESOURCES:**

AETECH *"Ada Training Environment"* and *"IntegrAda"* with *"On-Line Training and Reference Module"*.

VII. **PRESENTATION**

A. Introduction

1. Introduce a football field using integer type to describe the yard markers. Discuss that a yard marker of 51 yards might be legal, but wouldn't make any sense.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
1. Derived Type	1a. Definition - brand new type formed from a previously declared type.
	1b. Done with keyword "new".
	1c. May have all operations of base type.
2. Type Conversion	2a. Derived type may be able to be converted back to its base type if mixing of types is required. This will allow comparisons and operations to be performed on the converted type.
	2b. Syntax uses parenthesis around old type, and new type proceeding it. Must be stored in object of new type. (i.e. in an assignment statement). `New_Type_Object:= New_Type(Old_Type_Object);`

LABORATORY EXPERIMENT

I. **BLOCK:** II - "Fundamentals of Ada Programming"

II. **UNIT:** J

III. **LAB NUMBER:** 13

IV. **LAB TITLE:** "Derived Types"

V. **STUDENT OBJECTIVES:** At the completion of this experiment, the student should be able to:

1. Write a procedure which declares and uses derived types.

VI. **REQUIRED MATERIALS:**

1. Note taking materials.
2. AETECH *"IntegrAda"* with *"On-Line Training and Reference Module"*.
3. Student Data Disk.

VII. **PROCEDURE**

1. Given the following type declarations:

   ```
   type SpeedLimitType is range 0..65;

   type DistanceType is digits(2) range
   0.00..520.00;
   -- (where 520.0 represents the maximum distance
   that a person may travel at the maximum speed
   limit in an 8 hour day)

   type HourType is range 0..8;
   ```

   Write a procedure which prompts the user to enter an expected average speed and the number of hours expected to be driven during a trip. The program should calculate and output to the screen the distance that the user could expect to travel. Save the program as LAB13.ADA.

2. Compile, debug, bind, and execute the program.
3. Print out a copy of your program, and your executable output to turn in to your Instructor.
4. Power down computer, and clean up area.

```
                 --*************************--;
                 --*    Derived Types     *--;
                 --*************************--;

   -- Author's Name           : TEACHER GUIDE ;
   -- Assignment Number       : LAB # II.J ;

--------------------  Program Executive  ------------
   -- Below is a solution for Lab # II.J. This solution may
   -- be used by the instructor as a guide for helping
   -- students complete the laboratory assignment.
-------------------------------------------------------
   with TEXT_IO;  use TEXT_IO;

   procedure Calc_Distance is

      type SpeedLimitType is range 0..65;

      type DistanceType is digits( 2 ) range 0.00..520.00;

      type HourType is range 0..8;

      Avg_Speed : SpeedLimitType;
      Num_Hours : HourType;
      Distance  : DistanceType;

      package DistanceIO is new FLOAT_IO( DistanceType );

      package HourIO is new INTEGER_IO( HourType );

      package SpeedIO is new INTEGER_IO( SpeedLimitType );

   begin
      put( "Please enter your expected average speed: " );
      SpeedIO.get( Avg_Speed );
      NEW_LINE;   NEW_LINE;
      put( "Please enter your expected number of hours
            driving: " );
      HourIO.get( Num_Hours );
      NEW_LINE;   NEW_LINE;   NEW_LINE;

      Distance := DistanceType( Avg_Speed ) * DistanceType(
      Num_Hours );

      put( "You can expect to travel " );
      DistanceIO.put( Distance, Aft => 2, Exp => 0 );
      put( " miles." );
      NEW_LINE;   NEW_LINE;

   end CALC_DISTANCE;
```

INFORMATION LESSON PLAN

I. **BLOCK:** II - "Fundamentals of Ada Programming"

II. **UNIT:** K

III. **LESSON TITLE:** "Subtypes"

IV. **LESSON OBJECTIVES:** At the completion of this lesson, the student should be able to:

1. Define a subtype, and compare and contrast subtypes to derived types.
2. Discuss range constraints and accuracy constraints of subtypes.
3. Declare subtypes, and objects of type subtype.

V. **LEARNING ACTIVITIES:**

1. Take notes on lecture presented by Instructor.
2. Participate in class discussion of presented lecture.
3. CAI Assignment - Block III
   AETECH *"Ada Training Environment"* or *"IntegrAda"* with *"On-Line Training and Reference Module"*.

   Read & take notes on the following sections:

   Block III, Lesson 5, Topics 3-6.

   a. Subtypes.
   b. Range constraints.
   c. Accuracy constraints.
   d. Index changes.

VI. **SPECIAL RESOURCES:**

AETECH *"Ada Training Environment"* and *"IntegrAda"* with *"On-Line Training and Reference Module"*.

Skansholm, Ada from the Beginning, Addison-Wesley, 1988, pg. 105.

VII. PRESENTATION

A. Introduction

1. Discuss altitude for a plane, and identify how the set of integers (negative integers) wouldn't apply to altitude unless plane had crashed. Review Natural and Positive and show how these are subtypes of Integer.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
1. Subtypes	1a. Definition - a possible smaller range of a declared type. 1b. May be mixed with base type, thus saving memory and allowing for faster execution. 1c. Can place additional range constraints on base type. 1d. Does not increase accuracy. 1e. Useful for unconstrained arrays.
2. Predefined Subtypes	2a. Natural (0..Integer'Last). 2b. Positive (1..Integer'Last).

LABORATORY EXPERIMENT

I. **BLOCK:** II - "Fundamentals of Ada Programming"

II. **UNIT:** K

III. **LAB NUMBER:** 14

IV. **LAB TITLE:** "Subtypes"

V. **STUDENT OBJECTIVES:** At the completion of this experiment, the student should be able to:

1. Create and use a simple procedure in which subtypes are declared.

VI. **REQUIRED MATERIALS:**

1. Note taking materials.
2. AETECH *"IntegrAda"* with *"On-Line Training and Reference Module"*.
3. Student Data Disk.

VII. **PROCEDURE**

1. Write a procedure which declares the following subtypes and objects:

```
Subtype Letter_Grade_Type is Character range
'A'..'E';
Subtype Passing_Grade_Type is Letter_Grade_Type
   range 'A'..'D';
Subtype Num_Grade_Type is Integer range 0..100;
   Input_Grade:Num_Grade_Type;
   Letter_Grade:Letter_Grade_Type;
```

Have the procedure first prompt the user for a numeric grade from 0 through 100. When entered, the number is evaluated and the appropriate letter grade is assigned to the object Letter_Grade according to the following scale:

90 - 100 - A
80 - 89 - B
70 - 79 - C
60 - 69 - D
less than 60 - E

VII. PROCEDURE
(continued)

Have the procedure output the entered number along with the appropriate letter grade. Follow the example below:

```
Enter a Number (0 - 100): 87

            Entered Number = 87
             Letter Grade   = B
```

Save this program as LAB14.ADA.

2. Compile, debug, bind, and execute the program.
3. Print out a copy of your program, and your executable output to turn in to your Instructor.
4. Power down computer, and clean up area.

```
                   --******************--;
                   --*    Subtypes    *--;
                   --******************--;

  -- Author's Name           : TEACHER GUIDE ;
  -- Assignment Number       : LAB # II.K ;

 --------------------    Program Executive    ------------
  -- Below is a solution for Lab # II.K.  This solution may
  -- be used by the instructor as a guide for helping
  -- students complete the laboratory assignment.
 ----------------------------------------------------------
with TEXT_IO;    use TEXT_IO;

procedure CALC_GRADE is

   subtype Letter_Grade_Type is CHARACTER range 'A'..'E';
   subtype Passing_Grade_Type is Letter_Grade_Type range
   'A' .. 'D';
   subtype Num_Grade_Type is INTEGER range 0..100;

   Input_Grade : Num_Grade_Type;
   Letter_Grade : Letter_Grade_Type;

   package GradeIO is new INTEGER_IO( Num_Grade_Type );
   use GradeIO;

  begin
   put( "Please enter a numeric grade (between 0 and
   100):  ");
     get( Input_Grade );
     NEW_LINE;  NEW_LINE;  NEW_LINE;

     if ( Input_Grade < 60 ) then
         Letter_Grade := 'E';
     elsif ( Input_Grade < 70 ) then
        Letter_Grade := 'D';
     elsif ( Input_Grade < 80 ) then
        Letter_Grade := 'C';
     elsif ( Input_Grade < 90 ) then
        Letter_Grade := 'B';
     else
        Letter_Grade := 'A';
     end if;

     put( "    Entered Number = " );
     put( Input_Grade, Width => 1 );
     NEW_LINE;
     put("    Letter Grade    = " );
     put( Letter_Grade );
     NEW_LINE;
  end CALC_GRADE;
```

INFORMATION LESSON PLAN

I. **BLOCK:** II - "Fundamentals of Ada Programming"

II. **UNIT:** L

III. **LESSON TITLE:** "Subprograms"

IV. **LESSON OBJECTIVES:** At the completion of this lesson, the student should be able to:

1. Identify the two types of subprogram structures.
2. Identify proper techniques for naming subprograms.
3. Compare and contrast functions and procedures.
4. Identify the modes of subprogram parameters.
5. Write simple procedures and functions.
6. Call procedures and functions.
7. Define formal and actual parameters.
8. Assign objects to formal and actual parameters.
9. Understand the use of local variables.

V. **LEARNING ACTIVITIES:**

1. Take notes on lecture presented by Instructor.

2. Participate in class discussion of presented lecture.

3. CAI Assignment - Block IV
   AETECH *"Ada Training Environment"* or *"IntegrAda"* with *"On-Line Training and Reference Module"*.

   Read & take notes on the following sections:

   Block IV, Lesson 2, Topics 1-10.

   a. Description.
   b. Defining subprograms.
   c. Invoking subprograms.
   d. Parameters.
   e. Formal and actual parameters.
   f. Specifications.
   g. Matching parameters.
   h. Notational assignment.
   i. Bodies.
   j. Summary.

VI. SPECIAL RESOURCES:

AETECH *"Ada Training Environment"* and *"IntegrAda"* with *"On-Line Training and Reference Module"*.

Skansholm, Ada from the Beginning, Addison-Wesley, 1988, Ch. 6, pp. 215-276.

VII. PRESENTATION

A. Introduction

1. Discuss modularity and how programs should be designed using a top-down approach (written into the smallest programming units) , then introduce subprograms.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
1. Subprogram Structure	1a. Two types; functions and procedures. 1b. Functions must return a value procedures don't have to. 1c. Both can have parameters; functions only of mode "in". 1d. Both aid modularity 1e. Both must be declared. 1f. Both have bodies.
2. Parameter Modes	2a. Three types; **in, out, in out** **in** - comes into sub from caller. Value cannot be changed (value is the same after the call). **out** - sub creates some value (no value when it comes in, or garbage value), and returns value to where it was called. **in out** - items are passed into a subprogram. Items can be modified by subprogram, and items are then passed back to caller.
3. Parameter Types	3a. Two types: formal and actual formal - ones in specifications. actual - ones in call. 3b. Actual values are assigned to formal values when they are passed into sub. The sub, when completed, passes the formal parameters back out where they assume their actual names.
4. Syntax	4a. Both may have parameter specification. 4b. Both have **begin** and **end**.

LABORATORY EXPERIMENT

I. **BLOCK:** II - "Fundamentals of Ada Programming"

II. **UNIT:** L

III. **LAB NUMBER:** 15

IV. **LAB TITLE:** "Subprograms"

V. **STUDENT OBJECTIVES:** At the completion of this experiment, the student should be able to:

1. Write a simple function.
2. Write a simple procedure.

VI. **REQUIRED MATERIALS:**

1. Note taking materials.
2. AETECH *"IntegrAda"* with *"On-Line Training and Reference Module"*.
3. Student Data Disk.

VII. **PROCEDURE**

1. Write a function subprogram resource which calculates the factorial of an integer value.
   Example : 4! = 24.

```
                Use the following structure:
                :::
                begin
                  if value = 1 then
                      return 1 ;
                else
                  return Value * Factorial(Value - 1);
                end if;
       end Factorial;
```

   Note: This is called a recursive function.
   Save this as LAB 15A.ADA.

2. Compile this subprogram.

VII. PROCEDURE
(Continued)

3. Write a procedure subprogram resource which calculates the area of a rectangle, given the height, and width as integers.

   Save this as LAB15B.ADA.

4. Print out a copy of both LAB15A.ADA and LAB15B.ADA source code to turn in to your Instructor.

5. Power down computer and clean up area.

```
              --*********************--;
              --*    Subprograms   *--;
              --*********************--;

 -- Author's Name          : TEACHER GUIDE ;
 -- Assignment Number      : LAB # II.L ;

---------------------   Program Executive   -------------
 -- Below is a solution for Lab # II.L.  This solution may
 -- be used by the instructor as a guide for helping
 -- students complete the laboratory assignment.
--------------------------------------------------------
 function Factorial ( Value: INTEGER ) return INTEGER is
 begin
    if ( Value = 1 ) then
       return 1;
    else
       return Value * Factorial( Value - 1 );
    end if;
 end Factorial;
```

```
 procedure Calc_Area ( Height, Width : in INTEGER;
                       Area :out INTEGER ) is
 begin
    Area := Width * Height;
 end Calc_Area;
```

INFORMATION LESSON PLAN

I. **BLOCK:** II - "Fundamentals of Ada Programming"

II. **UNIT:** M

III. **LESSON TITLE:** "Packages"

IV. **LESSON OBJECTIVES:** At the completion of this lesson, the student should be able to:

1. Define package.
2. List and describe the two compilation units that make up a package.
3. Identify the two parts of a package specification.
4. Identify the two parts of a package body.
5. Define elaboration.
6. List the three logical ways to group programming resources into packages.

V. **LEARNING ACTIVITIES:**

1. Take notes on lecture presented by Instructor.
2. Participate in class discussion of presented lecture.
3. CAI Assignment - Block IV
   AETECH *"Ada Training Environment"* or *"IntegrAda"* with *"On-Line Training and Reference Module"*.

   Read & take notes on the following sections:

   Block IV, Lesson 1, Topics 1-10.

   a. Description.
   b. Example of a package.
   c. Package specification.
   d. Package body.
   e. Package body (Cont.).
   f. Package designs.
   g. Object oriented designs.
   h. Shared data packages.
   i. Abstract state machines.
   j. Summary.

VI. SPECIAL RESOURCES:

AETECH *"Ada Training Environment"* and *"IntegrAda"* with *"On-Line Training and Reference Module"*.

Skansholm, Ada from the Beginning, Addison-Wesley, 1988, Ch. 8, pp.341-376.

VII. PRESENTATION

A. Introduction

1. Discuss that we have thus far used only the resources contained in existing packages, and that we can actually write our own packages.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
1. Packages	1a. Definition - group of logically related software entities. 1b. Two parts; specification and body. Specification - tells what the package does Body - implements the functions that the specification describes.
2. Package Specification	2a. Could be divided into two parts; Visible - part which user can freely use. Hidden - (done via private or limited private types) no immediate access for user; done so that user cannot alter certain package items (restricts access).
3. Package Body	3a. May have a declarative section prior to executable code. 3b. Contains code that makes package perform as the specification describes.
3. Package Body (continued)	3c. If specification has procedure or function then those subprogram bodies appear here. 3d. May be compiled separately from package specification.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
4. Elaboration	4a. Done via the elaboration of a "with" clause. Makes package visible and the resources in the package usable.
5. Package Designs	5a. Three ways to group program elements together into a package; Object Oriented Design - groups objects together. Shared Data - groups data structures together (sorting etc) May have generics. Abstract State Machines - contains specification which tells of certain conditions and elements within package. (on, off, etc)

LABORATORY EXPERIMENT

I. **BLOCK:** II - "Fundamentals of Ada Programming"

II. **UNIT:** M

III. **LAB NUMBER:** 16

IV. **LAB TITLE:** "Creating Simple Packages"

V. **STUDENT OBJECTIVES:** At the completion of this experiment, the student should be able to:

1. Create a simple package consisting of a procedure and function.

VI. **REQUIRED MATERIALS:**

1. Note taking materials.
2. AETECH *"IntegrAda"* with *"On-Line Training and Reference Module"*.
3. Student Data Disk.

VII. **PROCEDURE**

1. Using the two subprograms created in Lab 15 (LAB15A.ADA and LAB15B.ADA) incorporate these subprograms into a package called MATHPKG.ADA. Compile this package, and print out a copy of your source code to turn in to your Instructor.
2. Write a simple procedure to "drive" your math package. The procedure should prompt the user to enter an integer, and then calculate the factorial of the input integer. In addition, the procedure should prompt the user to enter a height, and width for a rectangle, and then calculate the area of the rectangle. Save this program as Lab16.ada.
3. Compile, debug, bind, and execute Lab16.ada.
4. Print out a copy of your package and driver, and a copy of your executable output to turn in to your Instructor.
5. Power down computer, and clean up area.

```
                   --*******************--;
                   --*     Packages    *--;
                   --*******************--;

  -- Author's Name          : TEACHER GUIDE ;
  -- Assignment Number      : LAB # II.M ;

 --------------------   Program Executive   -------------
  -- Below is a solution for Lab # II.M.  This solution may
  -- be used by the instructor as a guide for helping
  -- students complete the laboratory assignment.
 -------------------------------------------------------
  package Math_Functions is
     function Factorial ( Value : INTEGER ) return INTEGER;
     procedure Calc_Area ( Height, Width : in INTEGER;
                           Area : out INTEGER );
  end Math_Functions;

  package body Math_Functions is

     function Factorial ( Value: INTEGER ) return INTEGER
is
     begin
        if ( Value = 1 ) then
           return 1;
        else
           return Value * Factorial( Value - 1 );
        end if;
     end Factorial;
      procedure Calc_Area ( Height, Width : in INTEGER;
                            Area: out INTEGER ) is
     begin
        Area := Width * Height;
     end Calc_Area;
   end Math_Functions;
 --*********************************************--;
 --*     Driver for Package Math Functions     *--;
 --*********************************************--;

with TEXT_IO, Math_Functions;
use TEXT_IO, Math_Functions;

procedure Test_Functions is

   Height, Wide, Area : INTEGER;
   Num, Result : INTEGER;

   package IntegerIO is new INTEGER_IO( INTEGER );
   use IntegerIO;

begin
   put_line( "This program tests the Math_Functions
   package." );
   put_line( "First, the Factorial of a given integer
   will be calculated." );
   put( "Please enter the desired integer value: ");
   get( Num );
   NEW_LINE;  NEW_LINE;
   Result := Factorial( Num );
   put( "The Factorial of " );
   put( Num, Width => 1 );
   put( " is " );
   put( Result, Width => 1 );
   NEW_LINE; NEW_LINE; NEW_LINE;
   put_line( "Now, the Area of a given rectangle will be
   calculated." );
   put( "Please enter the Width of a rectangle: " );
   get( Wide );
   NEW_LINE;
   put( "Please enter the Height of a rectangle: " );
   get( Height );
   NEW_LINE;  NEW_LINE;
   Calc_Area( Height, Wide, Area );
   put( "The Area of the given rectangle is: " );
   put( Area, Width => 1 );
   put_line( " Square Feet.");
   NEW_LINE;
 end Test_Functions;
```

INFORMATION LESSON PLAN

I. **BLOCK:** II "Fundamentals of Ada Programming"

II. **UNIT:** N

III. **LESSON TITLE:** "Declaring Subprograms and Creating Packages"

IV. **LESSON OBJECTIVES:** At the completion of this lesson, the student should be able to:

1. Create a simple subprogram.
2. Understand how to compile a subprogram and have a main procedure import it.
3. Identify the two parts of a package.
4. Create a simple package.

V. **LEARNING ACTIVITIES:**

1. Take notes on lecture presented by Instructor.
2. Participate in class discussion of presented lecture.
3. CAI Assignment - Block II
   AETECH *"Ada Training Environment"* or *"IntegrAda"* with *"On-Line Training and Reference Module"*.

   Read & take notes on the following sections:

   Block II, Lesson 1, Topics 7-9
   a. Declaring subprograms.
   b. Creating packages.
   c. Summary.

VI. **SPECIAL RESOURCES:**

AETECH *"Ada Training Environment"* and *"IntegrAda"* with *"On-Line Training and Reference Module"*.

VII. PRESENTATION

A. Introduction

1. Tell students that a large program (such as the one that controls the space shuttle) is coded by many programmers; therefore each programmer only programs a small block of code, and these small blocks (mostly subprograms) are put together to form the program. This is why we use specifications (which link the various blocks together).

B. Instructional Topics and Key Points

TOPIC	KEY POINT
1. Subprogram	1a. Definition - One small part of an entire system. Consists of functions and procedures. 1b. Two parts to a subprogram: a. Specification - interface. b. Body - actual implementation. 1c. Main Procedure - special form of procedure, runs on the operating system, and is said to "drive" system.
2. Package	2a. Definition - a group of logically related entities. 2b. Consist of two parts: a. Specification - interface. b. Body - actual implementation.
3. Declaring Subprogram	3a. Show how an internal subprogram is declared in a simple procedure.

LABORATORY EXPERIMENT

I. **BLOCK:** II - 'Fundamentals of Ada Programming"

II. **UNIT:** N

III. **LAB NUMBER:** 17

IV. **LAB TITLE:** "Declaring Subprograms and Creating Packages"

V. **STUDENT OBJECTIVES:** At the completion of this experiment, the student should be able to:

1. Create and use a package of mathematical subprogram resources.

VI. **REQUIRED MATERIALS:**

1. Note taking materials.
2. AETECH *"IntegrAda"* with *"On-Line Training and Reference Module"*.
3. Student Data Disk.

VII. **PROCEDURE**

1. Add to your math package created in Laboratory 16, which contains a procedure resource to calculate the area of a triangle, and a function resource to calculate square roots. Develop a package driver program which prompts the user to enter 3 integer values which are assigned as the sides of the triangle. If the entered sides are valid sides of a triangle, then return the area of the triangle; otherwise return a message and allow for reentry of valid sides. Valid sides are when the sum of any two sides are greater then the third. Area is computed with the formula:

$$\text{Area} = \sqrt{s(s-a)(s-b)(s-c)}$$

where
$s = (a+b+c)/2$

Note: You will have to convert the input sides to floating point numbers, and develop and use a Square Root function from this math library package.

VII. **PROCEDURE**
(Continued)

1. To decide whether input sides are legitimate sides of a triangle, please use the following logic:

```
if s1+s2>s3 and s2+s3>s1 and s1+s3>s2 then
    --find area;
 else
   put_line ("Invalid Sides");
end if;
```

2. Compile and debug your math package, saving it again as MATHPKG.ADA.

3. Compile, debug, bind, and execute your driver program, saving it as LAB17.ADA.

4. Print out copies of package and driver, and executable code to turn in to your Instructor.

5. Power down computer, and clean up area.

```
-*****************************************************--;
-*    Declaring Subprograms and Creating Packages    *--;
-*****************************************************--;

 -- Author's Name          : TEACHER GUIDE ;
 -- Assignment Number      : LAB # II.N ;

 -------------------   Program Executive   --------------
 -- Below is a solution for Lab # II.N.  This solution may
 -- be used by the instructor as a guide for helping
 -- students complete the laboratory assignment.
 ---------------------------------------------------------
 package MathPkg is
    function Factorial ( Value : INTEGER ) return INTEGER;
    procedure Calc_Rec_Area ( Height, Width : in INTEGER;
 Area : out INTEGER );
    function Sqrt ( Num : FLOAT ) return FLOAT;
    procedure Calc_Tri_Area ( Side1, Side2, Side3 : in
                              INTEGER;  Area : out FLOAT );
 end MathPkg;
  package body MathPkg is
    function Factorial ( Value: INTEGER ) return INTEGER is
    begin
       if ( Value = 1 ) then
          return 1;
       else
          return Value * Factorial( Value - 1 );
       end if;
    end Factorial;
     procedure Calc_Rec_Area ( Height, Width : in INTEGER;
                              Area : out INTEGER ) is
    begin
       Area := Width * Height;
    end Calc_Rec_Area;

    --   Approximate square root, using Newton's method:
    --   If you have a package which provides for a SQRT
    --   Function, you may wish to use that, in lieu of
    --   this solution.
    function Sqrt( Num : FLOAT ) return FLOAT is
       Root : Float  :=  Num / 2.0;
    begin
       while ( abs( Num - Root ** 2 ) > 2.0 * Num *
 Float'Epsilon ) loop
           Root := ( Root + Num / Root ) / 2.0;
       end loop;
       return Root;
    end Sqrt;
```

```
   procedure Calc_Tri_Area ( Side1, Side2, Side3 : in
                             INTEGER;
                             Area : out FLOAT ) is
      Sum, Temp_Area, A, B, C : FLOAT;
   begin
      A := FLOAT ( Side1 );
      B := FLOAT ( Side2 );
      C := FLOAT ( Side3 );
      Sum := ( A + B + C ) / 2.0;
      Temp_Area := Sum * ( Sum - A ) * ( Sum - B ) *
                   ( Sum - C );
      Area := Sqrt( Temp_Area );
   end Calc_Tri_Area;

end MathPkg;
```

```
  --********************************************--;
  --*          Driver for MathPkg              *--;
  --********************************************--;

with TEXT_IO, MathPkg;
use TEXT_IO, MathPkg;

procedure Triangles is

   Side1, Side2, Side3 : INTEGER;
   Area : FLOAT;
   Valid : Boolean;
   package IntegerIO is new INTEGER_IO( INTEGER );
   use IntegerIO;
   package FloatIO is new FLOAT_IO( FLOAT );
   use FloatIO;

begin
   put_line( "This program will calculate the area of a
   given triangle." );
   put_line( "Please enter INTEGER values when lengths
   are requested." );
   Valid := FALSE;
   while ( not Valid ) loop
      NEW_LINE;
      put( "Please enter length of side one: ");
      get( Side1 );
      NEW_LINE;
      put( "Please enter length of side two: ");
      get( Side2 );
      NEW_LINE;
      put( "Please enter length of side three: ");
      get( Side3 );
      NEW_LINE;  NEW_LINE;
      if ( Side1 + Side2 > Side3 ) and ( Side2 + Side3 >
           Side1 )
         and ( Side1 + Side3  > Side2 ) then
            Valid := TRUE;
            Calc_Tri_Area( Side1, Side2, Side3, Area );
      else
         put_line( "Invalid Sides!  Try Again..." );
      end if;
   end loop;
   put( "The Area of the given Triangle is: " );
   put( Area, Aft => 2, Exp => 0 );
   put_line( " Square Feet." );
   NEW_LINE;
end Triangles;
```

INFORMATION LESSON PLAN

I. **BLOCK:** II - "Fundamentals of Ada Programming"

II. **UNIT:** O

III. **LESSON TITLE:** "Ada Language Syntax"

IV. **LESSON OBJECTIVES:** At the completion of this lesson, the student should be able to:

1. Identify what limitations upper and lower case letters have on Ada syntax.
2. Define identifiers and discuss their limitations.
3. Define and identify numeric literals.
4. Define and identify character literals.
5. Define and identify string literals.
6. Define and identify the following delimiters: (from the Ada Language Reference Manual, section 2.2)

   ; : ' () . => .. :=

7. Define reserved word and identify their limitations.
8. Define program documentation, and discuss its importance for good programming practice.

V. **LEARNING ACTIVITIES:**

1. Take notes on lecture presented by Instructor.
2. Participate in class discussion of presented lecture.
3. CAI Assignment - Block II
   AETECH *"Ada Training Environment"* or *"IntegrAda"* with *"On-Line Training and Reference Module"*.

   Read & take notes on the following sections:

   Block II, Lesson 2, Topics 1-10

   a. Character Set.
   b. Lexical Units.
   c. Identifiers.
   d. Numeric Literals.
   e. Character Literals.
   f. Strings
   g. Simple Declarations.
   h. Compound Delimiters.
   i. Reserved Words.
   j. Comments.

VI. SPECIAL RESOURCES:

AETECH *"Ada Training Environment"* and *"IntegrAda"* with *"On-Line Training and Reference Module"*.

Skansholm, Ada From The Beginning, Addison-Wesley, 1988.

VII. PRESENTATION

A. Introduction

1. Discuss how our English symbols make our language understandable to us.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
1. Syntax for upper and lower case	1a. Compiler will not distinguish between upper and lower case because Ada is a non-case sensitive language. 1b. Conventional to put reserved words in lower case.
2. Identifier	2a. Can be used to name variable objects, data structures, program units, constants, exceptions, etc. 2b. Start with letter followed by letters, numbers, or single embedded underscores. 2c. Must fit on one line.
3. Numeric literals	3a. Are #'s either exact or real. 3b. Can use single embedded underscores without any effect (improves readability).
4. Character Literal	4a. Define ASCII. Also note package ASCII is inside package STANDARD. 4b. Any ASCII character enclosed by apostrophes.
5. Strings	5a. Define Strings. (Unconstrained array of characters) 5b. Identify string type and discuss the need to provide the string length at the time of object declaration.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
6. Delimiters	6a. Discuss the uses of the following delimiters: ; end of line. : type declaration. , separates two objects in type declaration. ' attributes. . field identifier. => is equal to. .. range. := assignment.
7. Reserved Words	7a. Define. 7b. Give handout with all reserved words on them and review each reserved word discussed to date (get handout from Ada Language Reference Manual 2.9)
8. Comments	8a. Discuss the importance of good programming documentation. 8b. Discuss how to comment.

LABORATORY EXPERIMENT

I. **BLOCK:** II - "Fundamentals of Ada Programming"

II. **UNIT:** O

III. **LAB NUMBER:** 18

IV. **LAB TITLE:** "Ada Language Syntax: Using Comments"

V. **STUDENT OBJECTIVES:** At the completion of this experiment, the student should be able to:

1. Provide meaningful comments within Ada programs.

VI. **REQUIRED MATERIALS:**

1. Note taking materials.
2. AETECH *"IntegrAda"* with *"On-Line Training and Reference Module"*.
3. Student Data Disk.

VII. **PROCEDURE**

1. Using the math package and driver created in Laboratory Experiment 17, edit both the package and the driver incorporating meaningful comments within each so that a user will be able to understand the operation of your package and driver. Save the package as MATHPKG.ADA and your driver as LAB18.ADA.
2. Recompile, debug, bind, and execute the driver prior to recompiling the math package and see what happens.
3. Print out a copy of your program, and your executable output to turn in to your Instructor.
4. Power down computer, and clean up area.

```
       --**********************************************--;
       --*    Ada Language Syntax: Using Comments     *--;
       --**********************************************--;

-- Author's Name          : TEACHER GUIDE ;
-- Assignment Number      : LAB # II.O ;

-- The quality and quantity of comments will of course vary
-- from student to student, but a reasonable collection
-- might include the following:

-- In the package:
     -- In function Factorial:
        -- Mention that this is a recursive function.
        -- Explain that Factorial( X ) is X * X-1 * X-2 *
        -- ... * 3 * 2 * 1.
        -- In function Sqrt:
          -- Explain (or at least mention) the method used.
        -- In procedure Calc_Tri_Area:
          -- Explain the need to convert lengths to
          -- floating point values.
          -- Possibly point out the usage of package
          -- function Sqrt.

    -- In the driver program:
      -- State that program calculates the area of a
      -- triangle.
      -- Explain the determination of a valid triangle.
      -- State that user is repeatedly prompted for lengths
      -- of sides until a valid triangle is obtained.
      -- Point out the usage of package procedure (
      -- Calc_Tri_Area ).
```

INFORMATION LESSON PLAN

I. **BLOCK:** II - "Fundamentals of Ada Programming"

II. **UNIT:** P

III. **LESSON TITLE:** "The 'If' Control Structure"

IV. **LESSON OBJECTIVES:** At the completion of this lesson, the student should be able to:

1. Understand and develop a simple program using the if..then control structure.
2. Understand and develop a simple program using an if.. then..else control structure.
3. Understand and develop a simple program using an if..then..elsif..else control structure.

V. **LEARNING ACTIVITIES:**

1. Take notes on lecture presented by Instructor.
2. Participate in class discussion of presented lecture.
3. CAI Assignment - Block II
   AETECH *"Ada Training Environment"* or *"IntegrAda"* with *"On-Line Training and Reference Module"*.

   Read & take notes on the following sections:

   Block II, Lesson 3, Topics 1-3

   a. if..then.
   b. if..then..else.
   c. if..then..elsif..else.

VI. **SPECIAL RESOURCES:**

AETECH *"Ada Training Environment"* and *"IntegrAda"* with *"On-Line Training and Reference Module"*.

Skansholm, Ada From The Beginning, Addison-Wesley, 1988, pp. 55-56.

VII. PRESENTATION

A. Introduction

1. Discuss why it is important for computers to be able to make choices or decisions based upon known conditions.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
1. If..then	1a. Used to determine whether an action will be taken. If the "if" part of the statement is true, then execute that code, otherwise ignore it. 1b. 3 Components of structure. a. if b. then c. end if 1c. No semicolon after then. 1d. Indentation for easier reading.
2. If..then..else	2a. Used to make a choice between 2 items. If the "if" part of statement is true then execute it, and ignore the "else" section; otherwise if the "if" part of the statement is false then ignore it, and execute the "else" section of the structure. 2b. 4 Components of structure: a. if b. then c. else d. end if 2c. Else not followed by semicolon.
3. If..then..elsif.. ..else	3a. Used to make a choice between 2 or more items. Provides for unlimited selection of action. A minimum of five components of the structure are required: a.If b.then c. else d. elsif e. endif 3b. Identify elsif spelling. 3c. Every elsif has a corresponding then.

LABORATORY EXPERIMENT

I. **BLOCK:** II - "Fundamentals of Ada Programming"

II. **UNIT:** P

III. **LAB NUMBER:** 19

IV. **LAB TITLE:** "The If.. Then Control Structure"

V. **STUDENT OBJECTIVES:**

1. The student will learn how to use simple If..Then control structures in Ada.

VI. **REQUIRED MATERIALS:**

1. Note taking materials.
2. AETECH *"IntegrAda"* with *"On-Line Training and Reference Module"*.
3. Student data diskette.

VII. **PROCEDURE:**

1. Write a procedure to prompt the user to enter 2 integers. If the 1st Integer entered is larger than the second integer entered output: "The first number is larger than the second number"

   Save this program as LAB19.ADA.

2. Compile, debug, bind, and execute the program.
3. Print out a copy of your program and output to turn in to your Instructor.
4. Power down computer, and clean up area.

```
        --*******************************************--;
        --*    The If...Then Control Structure    *--;
        --*******************************************--;

 -- Author's Name          : TEACHER GUIDE ;
 -- Assignment Number      : LAB # II.PA ;

------------------- Program Executive -------------
 -- Below is a solution for Lab # II.PA. This solution may
 -- be used by the instructor as a guide for helping
 -- students complete the laboratory assignment.
----------------------------------------------------------
 with TEXT_IO;  use TEXT_IO;

 procedure Larger is

    First, Second : INTEGER;

    package IntegerIO is new INTEGER_IO( INTEGER );
    use IntegerIO;

 begin
    put( "Please enter an integer value: " );
    get( First );
    NEW_LINE;
    put( "Now, please enter a second integer value: " );
    get( Second );
    NEW_LINE;  NEW_LINE;
    if ( First > Second ) then
       put( "The first number is larger than the second
             number." );
    end if;
 end Larger;
```

LABORATORY EXPERIMENT

I. **BLOCK:** II - "Fundamentals of Ada Programming"

II. **UNIT:** P

III. **LAB NUMBER:** 20

IV. **LAB TITLE:** "The If..Then..Elsif..Else Control Structure"

V. **STUDENT OBJECTIVES:** At the completion of this experiment, the student should be able to:

1. Create and use an If..Then..Elsif..Else Control Structure in Ada.

VI. **REQUIRED MATERIALS:**

1. Note taking materials.
2. AETECH *"IntegrAda"* with *"On-Line Training and Reference Module"*.
3. Student Data Disk.

VII. **PROCEDURE**

1. Write a main procedure which prompts the user to input a character. The program outputs whether the character entered was an upper case letter, a lower case letter, or not a letter at all. Utilize an If..Then..Elsif..Else Structure. Save the program as LAB20.ADA.
2. Compile, debug, bind, and execute the program.
3. Print out a copy of your program, and your executable output to turn in to your Instructor.
4. Power down computer, and clean up area.

```
-*****************************************************--;
-*    The If..Then..Elsif..Else Control Structure    *--;
-*****************************************************--;

  -- Author's Name          : TEACHER GUIDE ;
  -- Assignment Number      : LAB # II.PB ;

---------------   Program Executive   -------------------
-- Below is a solution for Lab # II.PB. This solution may
-- be used by the instructor as a guide for helping
-- students complete the laboratory assignment.
---------------------------------------------------------
  with TEXT_IO;  use TEXT_IO;

  procedure Check_Letter is

     Letter : CHARACTER;

  begin
     put( "Pick a character, any character... " );
     get( Letter );
     NEW_LINE;  NEW_LINE;
     if ( Letter in 'A'..'Z' ) then
        put( "Chosen character is an uppercase letter." );
     elsif ( Letter in 'a'..'z' ) then
        put( "Chosen character is a lowercase letter." );
     else
        put( "Chosen character is not a letter at all." );
     end if;
  end Check_Letter;
```

INFORMATION LESSON PLAN

I. **BLOCK:** II - "Fundamentals of Ada Programming"

II. **UNIT:** Q

III. **LESSON TITLE:** "The Case Control Structure"

IV. **LESSON OBJECTIVES:** At the completion of this lesson, the student should be able to:

1. Identify the four required components of a case structure.
2. Recognize that a case structure must have at least two alternatives.
3. Understand when to use a case structure.
4. Identify what types a case structure may be used with.
5. Understand the purpose of a null statement.
6. Write a program using the case structure.

V. **LEARNING ACTIVITIES:**

1. Take notes on lecture presented by Instructor.
2. Participate in class discussion of presented lecture.
3. CAI Assignment - Block II
   AETECH *"Ada Training Environment"* or *"IntegrAda"* with *"On-Line Training and Reference Module"*.

   Read & take notes on the following sections:

   Block II, Lesson 3, Topics 4, 10

   a. Case statement.
   b. Null statement.

VI. **SPECIAL RESOURCES:**

AETECH *"Ada Training Environment"* and *"IntegrAda"* with *"On-Line Training and Reference Module"*.

Volper, Katz, Introduction to Programming using Ada, Prentice-Hall, 1990, pp. 191-195.

VII. PRESENTATION

A. Introduction

1. Compare a case structure with a multiple choice structure on a test.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
1. Case Structure	1a. Four required components to a case structure: a. **case** b. **is** c. **when** d. **end case**
	1b. Used for multiple choice decisions related to a single variable.
	1c. Can only be used with discrete types (types with a known number of values). For types other than discrete, a case structure may not be utilized.
	1d. All possible values for variable must be accounted for.
	1e. Can accomplish 1d. above by using the "when others" option.
	1f. => means "if it is equal to".
2. Null Statement	2a. To satisfy 1d. above, sometimes it is necessary to use a null statement.
	2b. **Null** statement means no action will be taken (as it relates to case structures).

LABORATORY EXPERIMENT

I. **BLOCK:** II - "Fundamentals of Ada Programming"

II. **UNIT:** Q

III. **LAB NUMBER:** 21

IV. **LAB TITLE:** "The Case Control Structure"

V. **STUDENT OBJECTIVES:** At the completion of this experiment, the student should be able to:

1. Create and use a case control structure.

VI. **REQUIRED MATERIALS:**

1. Note taking materials.
2. AETECH *"IntegrAda"* with *"On-Line Training and Reference Module"*.
3. Student Data Disk.

VII. **PROCEDURE**

1. Write a program which prompts the user to input 5 integers, and prints to screen after each number has been entered whether the number is odd or even. Assume only odd or even numbers between 1 and 20 are evaluated. Use | Notation. Output variations should use the following: "Odd Number"; "Even Number"; "Number Out of Range". Use a case control structure. Save your program as LAB21.ADA.
2. Compile, debug, bind, and execute the program.
3. Print out a copy of your program, and your executable output to turn in to your Instructor.
4. Power down computer, and clean up area.

```
        --************************************--;
        --*    The Case Control Structure    *--;
        --************************************--;

 -- Author's Name          : TEACHER GUIDE ;
 -- Assignment Number      : LAB # II.Q ;

-------------------   Program Executive   --------------
 -- Below is a solution for Lab # II.Q.  This solution may
 -- be used by the instructor as a guide for helping
 -- students complete the laboratory assignment.
---------------------------------------------------------
with TEXT_IO;   use TEXT_IO;
procedure Try_Case is

   Num1, Num2, Num3, Num4, Num5 : INTEGER;

   package IntegerIO is new INTEGER_IO( INTEGER );
   use IntegerIO;

   procedure Odd_Or_Even( Num: in INTEGER ) is
   begin
      case Num is
         when  1 | 3 | 5 | 7 | 9 | 11 | 13 | 15 | 17 | 19
                 =>
            put_line( "Odd Number" );

         when  2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20
                 =>
            put_line( "Even Number" );

         when others  =>  put_line( "Number Out of Range"
         );
      end case;
      NEW_LINE;
   end Odd_Or_Even;

begin
   put_line( "Please enter Integer values in the range
              1..20." );
   NEW_LINE;
   put( "Enter first value: " );
   get( Num1 );
   Odd_Or_Even( Num1 );
   put( "Enter second value: " );
   get( Num2 );
   Odd_Or_Even( Num2 );
   put( "Enter third value: " );
   get( Num3 );
   Odd_Or_Even( Num3 );
   put( "Enter fourth value: " );
   get( Num4 );
   Odd_Or_Even( Num4 );
   put( "Enter fifth value: " );
   get( Num5 );
   Odd_Or_Even( Num5 );
end Try_Case;
```

INFORMATION LESSON PLAN

I. **BLOCK:** II - "Fundamentals of Ada Programming"

II. **UNIT:** R

III. **LESSON TITLE:** "The Loop Control Structure"

IV. **LESSON OBJECTIVES:** At the completion of this lesson, the student should be able to:

1. Understand when and how to use a loop structure.
2. Describe the differences between a loop..exit, for..loop, in reverse..loop, and while..loop.
3. Write a program using each of the structures in (2) above.

V. **LEARNING ACTIVITIES:**

1. Take notes on lecture presented by Instructor.
2. Participate in class discussion of presented lecture.
3. CAI Assignment - Block II
   AETECH *"Ada Training Environment"* or *"IntegrAda"* with *"On-Line Training and Reference Module"*.

   Read & take notes on the following sections:

   Block II, Lesson 3, Topics 5-8

   a. Loop..exit.
   b. For..loop.
   c. In reverse..loop.
   d. While..loop.

VI. **SPECIAL RESOURCES:**

AETECH *"Ada Training Environment"* and *"IntegrAda"* with *"On-Line Training and Reference Module"*.

Volper, Katz, Introduction to Programming using Ada, Prentice Hall, 1990, pp. 159.

VII. PRESENTATION

A. Introduction

1. Compare a loop structure with the countdown of the space shuttle launch.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
1. loop..exit	1a. Allows for multiple iteration of Ada program statements.
	1b. Termination of the loop occurs when the exit statement is reached, and program begins execution after the end loop statement.
	1c. Loops may be nested (one loop completely inside another).
	1d. Loops may be named.
2. For..loop	2a. Specialized loop which has a built in counter to count the number of times that the loop has iterated (i.e. an integer range like 1..10).
	2b. Index values exist inside loop, and do not exist outside of loop. Loop parameters cannot be altered inside loop.
	2c. For..loop structures can be named, and may contain exit statements.
3. In reverse..loop	3a. Same as for..loop only counts backwards. However, range is given forward (from smallest to largest).
4. While..loop	4a. Conditional loop; will loop as long as condition remains true. When condition is false, program execution begins after the end loop.

LABORATORY EXPERIMENT

I. **BLOCK:** II - "Fundamentals of Ada Programming"

II. **UNIT:** R

III. **LAB NUMBER:** 22

IV. **LAB TITLE:** "The Loop Control Structure"

V. **STUDENT OBJECTIVES:** At the completion of this experiment, the student should be able to:

1. Create and use a simple Ada loop structure, which contains an exit statement.

VI. **REQUIRED MATERIALS:**

1. Note taking materials.
2. AETECH *"IntegrAda"* with *"On-Line Training and Reference Module"*.
3. Student Data Disk.

VII. **PROCEDURE**

1. Create a procedure which uses a simple loop.. exit structure, which prompts the user to enter integers from the keyboard, and adds the input integers in an accumulator. The program quits when the accumulator equals or exceeds 100. The program should output to the monitor the value of the accumulator as it goes through the loop each time. When the value of the accumulator reaches or exceeds 100, the program should display "Normal Program Termination". Save your program as LAB22.ADA.
2. Compile, debug, bind, and execute the program.
3. Print out a copy of your program, and your executable output to turn in to your Instructor.
4. Power down computer, and clean up area.

```
          --************************************--;
          --*    The Loop Control Structure    *--;
          --************************************--;

-- Author's Name          : TEACHER GUIDE ;
-- Assignment Number      : LAB # II.RA ;

-----------------   Program Executive   -----------------
-- Below is a solution for Lab # II.RA. This solution may
-- be used by the instructor as a guide for helping
-- students complete the laboratory assignment.
---------------------------------------------------------
with TEXT_IO;  use TEXT_IO;

procedure Simple_Sum is

   Value, Sum : INTEGER;

   package IntegerIO is new INTEGER_IO( INTEGER );
   use IntegerIO;

begin
   Sum := 0;
   loop
      put( "Please enter an Integer value: " );
      get( Value );
      NEW_LINE;
      Sum := Sum + Value;
      put( "Current Value of Sum is: ");
      put( Sum, Width => 1 );
      NEW_LINE;  NEW_LINE;
      exit when Sum >= 100;
   end loop;
   put_line( "Normal Program Termination." );
end Simple_Sum;
```

LABORATORY EXPERIMENT

I. **BLOCK:** II - "Fundamentals of Ada Programming"

II. **UNIT:** R

III. **LAB NUMBER:** 23

IV. **LAB TITLE:** "Loop and Reverse Loop"

V. **STUDENT OBJECTIVES:** At the completion of this experiment, the student should be able to:

1. Write an Ada program which uses a simple For.. loop.
2. Write an Ada program which uses a reverse loop.

VI. **REQUIRED MATERIALS:**

1. Note taking materials.
2. AETECH *"IntegrAda"* with *"On-Line Training and Reference Module"*.
3. Student Data Disk.

VII. **PROCEDURE**

1. Write a procedure which prompts the user for a positive integer, and then outputs the summation of numbers from 1 to the input integer value. Save this program as LAB23A.ADA
2. Compile, debug, bind, and execute the program.
3. Print out a copy of your program, and your executable output to turn in to your Instructor.

4. Write a program which simulates a shuttle countdown starting at 10 seconds and going to zero. Output "lift-off" after countdown. Use a delay statement. Use a reverse loop. Output should be as follows:

   ```
       10
        9
        8
        7
        6
        5
        4
        3
        2
        1
   "LIFT - OFF"
   ```

   Save this program as LAB23B.ADA.

5. Follow steps 2 and 3 above.

6. Power down computer, and clean up area.

```
            --*******************************--;
            --*    Loop and Reverse Loop    *--;
            --*******************************--;

-- Author's Name          : TEACHER GUIDE ;
-- Assignment Number      : LAB # II.RB ;

-------------------   Program Executive   ----------------
-- Below is a solution for Lab # II.RB. This solution may
-- be used by the instructor as a guide for helping
-- students complete the laboratory assignment.
---------------------------------------------------------
with TEXT_IO;  use TEXT_IO;

procedure For_Sum is
   I, Limit, Sum : NATURAL;
   package NatIO is new INTEGER_IO( NATURAL );
   use NatIO;
begin
 put( "Enter a Positive Integer to serve as the limit of a
       summation: " );
   get( Limit );
   NEW_LINE;  NEW_LINE;
   Sum := 0;
   for I in 1 .. Limit loop
      Sum := Sum + I;
   end loop;
   put( "The Summation of all Integers from 1 to " );
   put( Limit, Width => 1 );
   put( " is: " );
   put( Sum, Width => 1 );
   NEW_LINE;
end For_Sum;
```

```
with TEXT_IO;  use TEXT_IO;
procedure CountDown is
   I : NATURAL;
   package NatIO is new INTEGER_IO( NATURAL );
   use NatIO;
begin
   put_line( "Countdown..." );
   NEW_LINE;
   for I in reverse 1 .. 10 loop
      put( I, Width => 3 );
      put_line( " ..." );
      delay( 1.0 );
   end loop;
   put_line( "  LIFT-OFF" );
end CountDown;
```

LABORATORY EXPERIMENT

I. **BLOCK:** II - "Fundamentals of Ada Programming"

II. **UNIT:** R

III. **LAB NUMBER:** 24

IV. **LAB TITLE:** "The While..Loop Control Structure"

V. **STUDENT OBJECTIVES:** At the completion of this experiment, the student should be able to:

1. Create and use a simple Ada procedure which uses a While..Loop structure.

VI. **REQUIRED MATERIALS:**

1. Note taking materials.
2. AETECH *"IntegrAda"* with *"On-Line Training and Reference Module"*.
3. Student Data Disk.

VII. **PROCEDURE**

1. Create a procedure which uses a while loop to input an undetermined number of student grades. The grades can have a value between 1 and 100 (range 1..100). Input these grades from the keyboard. The loop is terminated when a value outside the grade range is given. Finally, the procedure outputs to the screen the average of the grades entered. Save this program as LAB24.ADA.
2. Compile, debug, bind, and execute the program.
3. Print out a copy of your program, and your executable output to turn in to your Instructor.
4. Power down computer, and clean up area.

```
     --********************************************--;
     --*    The While..Loop Control Structure    *--;
     --********************************************--;

-- Author's Name          : TEACHER GUIDE ;
-- Assignment Number      : LAB # II.RC ;

------------------   Program Executive   ----------------
-- Below is a solution for Lab # II.RC. This solution may
-- be used by the instructor as a guide for helping
-- students complete the laboratory assignment.
---------------------------------------------------------
with TEXT_IO;  use TEXT_IO;
procedure Average_Grades is
   Grade, Sum, Num_Grades : INTEGER;
   Avg : FLOAT;
   package IntegerIO is new INTEGER_IO( INTEGER );
   use IntegerIO;
   package RealIO is new FLOAT_IO( FLOAT );
begin
   Sum := 0;
   Num_Grades := 0;
   put_line( "Please NOTE:  All Test Scores are to be in
   the range 1 .. 100.");
   put_line( "Enter a value outside that range
   to terminate." );
   NEW_LINE;  NEW_LINE;
   put( "Please Enter First Test Score: " );
   get( Grade );
   NEW_LINE;
   while ( Grade in 1 .. 100 ) loop
      Num_Grades := Num_Grades + 1;
      Sum := Sum + Grade;
      put( "Please Enter Next Test Score (negative or > 100
      to stop): ");
      get( Grade );
      NEW_LINE;
   end loop;
   NEW_LINE;  NEW_LINE;
   if ( Num_Grades > 0 ) then
      Avg :=  FLOAT ( Sum ) / FLOAT ( Num_Grades );
      put( "Average of " );
      put( Num_Grades, Width => 1 );
      put( " grades is: ");
      RealIO.put( Avg, Aft => 2, Exp => 0 );
      NEW_LINE;
   else
      put_line( "There is no Average because No Grades were
      Entered!!" );
   end if;
end Average_Grades;
```

INFORMATION LESSON PLAN

I. **BLOCK:** II - "Fundamentals of Ada Programming"

II. **UNIT:** S

III. **LESSON TITLE:** "Style"

IV. **LESSON OBJECTIVES:** At the completion of this lesson, the student should be able to:

1. Understand how a program's style makes the program more understandable.
2. Choose appropriate names for types and objects.
3. Choose appropriate names for packages.
4. Choose appropriate names for procedures and functions.
5. Understand the importance of indentation, and be able to indent the logical levels of a program.
6. Know when and when not to place a cr/lf in a program.

V. **LEARNING ACTIVITIES:**

1. Take notes on lecture presented by Instructor.
2. Participate in class discussion of presented lecture.
3. CAI Assignment - Block II
   AETECH *"Ada Training Environment"* or *"IntegrAda"* with *"On-Line Training and Reference Module"*.

   Read & take notes on the following sections:

   Block II, Lesson 4, Topics 1, 2, 5-10

   a. Precision in naming.
   b. Simple objects and types.
   c. Packages.
   d. Other program units.
   e. Logical indentation.
   f. Declaration/assignment alignment.
   g. Conditional blocks.
   h. Line spacing.

VI. SPECIAL RESOURCES:

AETECH *"Ada Training Environment"* and *"IntegrAda"* with *"On-Line Training and Reference Module"*.

VII. PRESENTATION

A. Introduction

1. Describe how hard it would be to read a book or text that just ran together, without any chapters, table of contents, etc. and compare this to a program without any style.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
1. Naming	1a. Names should be descriptive enough to allow another programmer to understand what the program is doing.
	1b. All lexical units must fit onto 1 line. (Block II, Lesson 2, Topic 2 for lexical information).
	1c. Again, Ada is non-case sensitive. It makes no difference to compiler whether names are in upper or lower case letters, or a combination of both. (Refer to mil-spec 1815a for required gov't style; also AETECH's "Integrada" for "Pretty Print").
2. Objects and Types	2a. Object or "box" must be declared to be of a certain type; box can only hold its own type of things.
	2b. Objects and object-types should be named using noun names. Type names should contain the word "type" at the end of their new name.
3. Naming Packages	3a. A package should have a name that conveys to the user what the package provides, thus naming of a package may require additional thought so that the user will know what tools are available within package.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
4. Naming procedures and functions	4a. Procedures perform some action; therefore, use imperative verb phrases when naming them, that represent some action.
	4b. Functions return a value; therefore, use a noun when naming them that represents the value.
5. Program Indentation	5a. Align a program so that it becomes more readable and understandable.
	5b. Main programming blocks (procedure, begin, end, etc) should stand out.
	5c. If structures, case structures, loops, etc should be indented to stand out. This will not only create more readable code, but also makes debugging the program easier.
	5d. Align colons, is, variables, etc. for better readability.
6. Line Spacing	6a. Use line spaces wherever it makes a program more readable. Don't put line spaces in between same kinds of declarations.

BLOCK III

BLOCK III

Advanced Ada Topics

INFORMATION LESSON PLAN

I. **BLOCK:** III - "Advanced Ada Topics"

II. **UNIT:** A

III. **LESSON TITLE:** "Type Attributes"

IV. **LESSON OBJECTIVES:** At the completion of this lesson, the student should be able to:

1. Define attribute.
2. Use the syntax for expressing an attribute.
3. Be able to identify and use the following attributes:

    'First
    'Last
    'Succ
    'Pred
    'Pos
    'Digits
    'Small
    'Large

V. **LEARNING ACTIVITIES:**

1. Take notes on lecture presented by Instructor.
2. Participate in class discussion of presented lecture.
3. CAI Assignment - Block III
   AETECH *"Ada Training Environment"* or *"IntegrAda"* with *"On-Line Training and Reference Module"*.

   Read & take notes on the following sections:

   Block III, Lesson 2, Topics 9-10

   a. Using attributes.
   b. Scalar attributes.

VI. **SPECIAL RESOURCES:**

AETECH *"Ada Training Environment"* and *"IntegrAda"* with *"On-Line Training and Reference Module"*.

Skansholm, Ada From the Beginning, Addison- Wesley, 1988.

VII. PRESENTATION

A. Introduction

1. Explain how attributes may be able to help a program become more readable and understandable. Enumerate days of week and show attributes of list.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
1. Attributes	1a. Definition - because scalar types are ordered sets of elements, attributes (relationships within list) may be able to be defined. An attribute is a characteristic of a value in a set. 1b. Syntax is to use an apostrophe, then (if required) other data in parenthesis.
2. Integer Attributes	2a **A'First** - yeilds the lower bound of A for a scalar. **A'Last** - yeilds upper bound of A for a scalar.
3. Float Attributes	3a. **'Digits** - Number of significant digits. **'Small** - smallest number that can be stored. **'Large** - largest number that can be stored.
4. Enumeration Attributes	4a. **'First** - first item in enumerated list. **'Last** - last item in enumerated list. **'Prec(Item)** - returns predecessor of ITEM in list. **'Succ(Item)** - returns successor of ITEM in list.
5. Character Attributes	5a. **'Pos(Char)** - gives position number of Char in ASCII table. **'Val(NUM)** - gives char. in Num position within ASCII table.

INFORMATION LESSON PLAN

I. **BLOCK:** III - "Advanced Ada Topics"

II. **UNIT:** B

III. **LESSON TITLE:** "More Attributes"

IV. **LESSON OBJECTIVES:** At the completion of this lesson, the student should be able to:

1. Identify what would be returned by the following attributes used with their appropriate types:

'Val(X)	**'Length**
'Range	**'Value(X)**
'Aft	**'Image(X)**
'Fore	**'Width**
'Delta	

V. **LEARNING ACTIVITIES:**

1. Take notes on lecture presented by Instructor.
2. Participate in class discussion of presented lecture.
3. CAI Assignment - Block V
   AETECH *"Ada Training Environment"* or *"IntegrAda"* with *"On-Line Training and Reference Module"*.

   Read & take notes on the following sections:

   Block V, Lesson 4, Topics 1-4, 6.

   a. Notation and use.
   b. Discrete types.
   c. Integers.
   d. Arrays.
   e. Floating/Fixed point.

VI. **SPECIAL RESOURCES:**

AETECH *"Ada Training Environment"* and *"IntegrAda"* with *"On-Line Training and Reference Module"*.

VII. **PRESENTATION**

A. Introduction

1. Discuss how using attributes can help in the describing of items, and make the finding/retrieving of items easier.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
1. List of Attributes	1a. **'VAL(X)** -returns value of data at position X.
	1b. **'Value(X)** - returns type value of X.
	1c. **'Image(X)** - returns string value of X. Returns string decimal value for integer X.
	1d. **'Width** - returns the longest value of X.
	1e. **'Length** - returns the number of items in list.
	1f. **'Range** - returns the range of the list from 'first..'last.
	1g. **'Aft** - returns the number of digits after decimal point.
	1h. **'Fore** - returns the number of digits before the decimal point (includes -sign).
	1i. **'Delta** - returns declared delta.
	1j. **'Digits** - returns the declared digits.

INFORMATION LESSON PLAN

I. **BLOCK:** III - "Advanced Ada Topics"

II. **UNIT:** C

III. **LESSON TITLE:** "Records"

IV. **LESSON OBJECTIVES:** At the completion of this lesson, the student should be able to:

1. Define record.
2. Declare a record type.
3. Declare objects of type record.
4. Assign components of a declared object of type record using dot notation, and either positional association, or named association.

V. **LEARNING ACTIVITIES:**

1. Take notes on lecture presented by Instructor.
2. Participate in class discussion of presented lecture.
3. CAI Assignment - Block II & III
   AETECH *"Ada Training Environment"* or *"IntegrAda"* with *"On-Line Training and Reference Module"*.

   Read & take notes on the following sections:

   Block II, Lesson 4, Topic 4.

   a. Records

   Block III, Lesson 4, Topics 1-3, 5, 6.

   a. Record types.
   b. Object declaration.
   c. Selected component notation.
   d. Aggregates.
   e. Composite types.

VI. **SPECIAL RESOURCES:**

AETECH *"Ada Training Environment"* and *"IntegrAda"* with *"On-Line Training and Reference Module"*.

Skansholm, Ada from the Beginning, Addison-Wesley, 1988, pg. 320.

VII. PRESENTATION

A. Introduction

1. Describe how a person's address is made up of different data types, and how records could be used to describe an address.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
1. Record Definition	1a. Group of possible heterogeneous (not of the same type) items. 1b. Records usually consists of components of different types which may be scalars, arrays, tasks, or other records.
2. Record Declaration	2a. type _____ is record <object declarations> end record; 2b. Indent for readability. 2c. end record has ;
3. Declaring objects of type Record	3a. Done as the declaration of other objects is done; only must come after the record type has been declared.
4. Component Assignment	4a. May be performed several ways: a. Dot notation - object.component_name followed by :=. b. positional association c. named association (most readable).

LABORATORY EXPERIMENT

I. **BLOCK:** III - "Advanced Ada Topics"

II. **UNIT:** C

III. **LAB NUMBER:** 25

IV. **LAB TITLE:** "Records"

V. **STUDENT OBJECTIVES:** At the completion of this experiment, the student should be able to:

1. Create and use a main Ada procedure which utilizes a record construct.

VI. **REQUIRED MATERIALS:**

1. Note taking materials.
2. AETECH *"IntegrAda"* with *"On-Line Training and Reference Module"*.
3. Student Data Disk.

VII. **PROCEDURE**

1. Create the following record types:

```
type Name_type is record
  L_Name: String(1..20);
  F_Name: String(1..20);
  MI : Character;
end record;

type St_Type is record
  Name: Name_type;
  Age : Integer;
  GPA : Float;
end record;
```

Declare 3 objects of type St_Type. Have the user enter the data for 3 students. Output the 3 students names, age, and GPA of each student. Calculate the average age, and average GPA of the three students and output this information.

Save this program as LAB25.ADA.

2. Compile, debug, bind, and execute the program.
3. Print out a copy of your program, and your executable output to turn in to your Instructor.
4. Power down computer, and clean up area.

```
                  --******************--;
                  --*     Records    *--;
                  --******************--;

-- Author's Name              : TEACHER GUIDE ;
-- Assignment Number          : LAB # III.C ;

-------------------   Program Executive   ---------------
-- Below is a solution for Lab # III.C. This solution may
-- be used by the instructor as a guide for helping
-- students complete the laboratory assignment.
---------------------------------------------------------
with TEXT_IO;  use TEXT_IO;
procedure Students is

  type Name_Type is
         record
           L_Name : STRING( 1..20 );
           F_Name : STRING( 1..20 );
           MI : CHARACTER;
         end record;

  type St_Type is
        record
          Name : Name_Type;
          Age  : INTEGER;
          GPA  : FLOAT;
        end record;

  St_1, St_2, St_3 : St_Type;
  Avg_Age, I : INTEGER;
  Avg_GPA : FLOAT;

  package IntegerIO is new INTEGER_IO( INTEGER );
  use IntegerIO;

  package FloatIO is new FLOAT_IO( FLOAT );
  use FloatIO;

procedure get_data( Student : out St_Type ) is
begin
  put( "     Last Name: ======> " );
  I := 0;
  while ( not End_of_Line ) loop
     I := I + 1;
     get( Student.Name.L_Name( I ) );
  end loop;
  Student.Name.L_Name(I + 1..20) := (I + 1..20 => ' ');
   SKIP_LINE;
   put( "     First Name: =====> " );
   I := 0;
  while ( not End_of_Line ) loop
      I := I + 1;
      get( Student.Name.F_Name( I ) );
  end loop;
  Student.Name.F_Name(I + 1..20 ) := (I + 1..20 => ' ');
```

```
    NEW_LINE;
    put( "      Middle Initial: => " );
    get( Student.Name.MI );
    NEW_LINE;
    put( "      Age: ============> " );
    get( Student.Age );
    NEW_LINE;
    put( "      GPA: ============> " );
    get( Student.GPA );
    NEW_LINE;  NEW_LINE;  SKIP_LINE;
  end get_data;
  procedure print_data( Student : in St_Type ) is
  begin
    put( Student.Name.L_Name );
    put( Student.Name.F_Name );
    put( Student.Name.MI );
    put( Student.Age, Width => 9 );
    put( Student.GPA, Fore => 3, Aft => 2, Exp => 0 );
    NEW_LINE;
  end print_data;

begin  -- procedure Students
   put_line( "Please enter the following information for
Student #1 :" );
   get_data( St_1 );
   put_line( "Please enter the following information for
Student #2 :" );
   get_data( St_2 );
   put_line( "Please enter the following information for
Student #3 :" );
   get_data( St_3 );
   NEW_LINE;
   put_line( "                         Summary of Student
Data: " );
   put_line( "-------------- Name ------------------");
   put_line( "     Last            First           Initial
   Age   GPA" );
   print_data( St_1 );
   print_data( St_2 );
   print_data( St_3 );
   NEW_LINE;   NEW_LINE;
   Avg_Age := ( St_1.Age + St_2.Age + St_3.Age ) / 3;
   put( "The Average Age of the Three Students is: ");
   put( Avg_Age, Width => 1 );
   NEW_LINE;  NEW_LINE;
   Avg_GPA := ( St_1.GPA + St_2.GPA + St_3.GPA ) / 3.0;
   put( "The Average GPA of the Three Students is: ");
   put( Avg_GPA, Aft => 2, Exp => 0 );
   NEW_LINE;
end Students;
```

INFORMATION LESSON PLAN

I. **BLOCK:** III - "Advanced Ada Topics"

II. **UNIT:** D

III. **LESSON TITLE:** "Arrays"

IV. **LESSON OBJECTIVES:** At the completion of this lesson, the student should be able to:

1. Define array.
2. Identify how to index an array.
3. Define unconstrained array, and identify the syntax for an unconstrained array.
4. Describe multidimensional arrays, and list an example of their use.
5. Assign components to a declared array using named and positional association.

V. **LEARNING ACTIVITIES:**

1. Take notes on lecture presented by Instructor.
2. Participate in class discussion of presented lecture.
3. CAI Assignment - Block II & III AETECH *"Ada Training Environment"* or *"IntegrAda"* with *"On-Line Training and Reference Module"*.

   Read & take notes on the following sections:

   Block II, Lesson 4, Topics 3

   a. Arrays

   Block III, Lesson 3, Topics 1-11

   a. Simple arrays.
   b. Indices.
   c. Unconstrained arrays.
   d. Multidimensional arrays.
   e. Operations with components.
   f. Assignments.
   g. Aggregates.
   h. Positional association.
   i. Named association.
   j. Aggregate Ranges.
   k. Initialization.

VI. SPECIAL RESOURCES:

AETECH *"Ada Training Environment"* and *"IntegrAda"* with *"On-Line Training and Reference Module"*.

P. Texel, Introductory Ada, Wadsworth Publishing, 1986, pp. 198-199.

VII. PRESENTATION

A. Introduction

1. Compare an egg box with a regular box, in that an egg box has several (12) different areas which store items of the same type (eggs). Then introduce arrays, and compare them to the egg box.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
1. Array Definition	1a. Group of homogeneous (of the same type) objects. 1b. Two types of arrays; constrained and unconstrained. constrained - known boundaries at time of type declaration. unconstrained - boundaries are not known at time of type declaration.
2. Indexing Arrays	2a. Definition - indexing is method of labeling each element in an array. Any discrete type may be used to index an array. Done inside parentheses.
3. Unconstrained Arrays	3a. Uses < > syntax to inform the compiler that bounds are not known at this time.
4. Multidimensional Arrays	4a. Arrays can have as many dimensions as required. Helps to better identify what's being programmed, leading to better understandability and readability.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
5. Assigning Array Components	5a. May be done several ways: 1. By index number 2. By slice (a range of indices). 3. By positional association (the position of the assignment items represent their assignment to array). 4. By named association (list name of array component, followed by => to the assignment component. 5. By a combination of above.

LAB III.D

LABORATORY EXERCISE

I. **BLOCK:** III - "Advanced Ada Topics"

II. **UNIT:** D

III. **LAB NUMBER:** 26

IV. **LAB TITLE:** "Arrays"

V. **STUDENT OBJECTIVES:** At the completion of this laboratory exercise, the student should be able to:

1. Create and use array type objects in an Ada main procedure.

VI. **REQUIRED MATERIALS**

1. Note taking materials.
2. AETECH *"IntegrAda"* with *"On-Line Training and Reference Module"*.
3. Student Data Disk.

VII. **PROCEDURE**

1. Write a procedure which declares two arrays with indices ranging from 1 to 10. Using a "for" loop, assign the components of the first array with the consecutive even numbers from 2 to 20. Using a simple loop, assign the components of the second array with the consecutive odd numbers from 1 to 19. Finally, using a "while" loop, output the following table of values:

```
INDEX    FIRST    SECOND    FIRST+SECOND    FIRST-SECOND

  1        2        1             3               1
  2        4        3             7               1
  3        6        5            11               1
 etc.     ...      ...           ...             ...
 10       20       19            39               1
```

   Save your program as LAB26.ADA.

2. Compile, bind, and execute the program.
3. Print out a copy of your source code and output to be turned in to your Instructor.
4. Power down computer, and clean work up area.

```
                    --*****************--;
                    --*    Arrays     *--;
                    --*****************--;

-- Author's Name          : TEACHER GUIDE ;
-- Assignment Number      : LAB # III.D ;

-------------------    Program Executive    ---------------
-- Below is a solution for Lab # III.D. This solution may
-- be used by the instructor as a guide for helping
-- students complete the laboratory assignment.
----------------------------------------------------------
with TEXT_IO;   use TEXT_IO;

procedure Try_Arrays is

   subtype Positions is POSITIVE range 1 .. 10;
   subtype Values is POSITIVE range 1 .. 20;

   Even_Array,
   Odd_Array : array ( Positions ) of Values;

   I : POSITIVE;

   package PosIO is new INTEGER_IO( POSITIVE );
   use PosIO;

begin
   for  I in Positions  loop
      Even_Array( I ) := I * 2;
   end loop;

-- The following loop could easily be incorporated into the
-- above loop
-- (for example, by adding the statement
-- Odd_Array( I ) := Even_Array( I ) - 1;
-- immediately before the end loop;).  The following
-- adheres to the Laboratory Exercise instructions.

  I := 1;
  loop
    Odd_Array( I ) := I * 2 - 1;
    exit when I = 10;
    I := I + 1;
  end loop;

  put_line("INDEX     FIRST     SECOND     FIRST + SECOND
FIRST - SECOND");
  NEW_LINE;
```

```
 -- The following loop could also be incorporated into the
 -- first loop above.

   I := 1;
   while ( I <= 10 ) loop
      put( I, Width => 3 );
      put( Even_Array( I ), Width => 10 );
      put( Odd_Array( I ), Width => 11 );
      put( Even_Array( I ) + Odd_Array( I ), Width => 14 );
      put( Even_Array( I ) - Odd_Array( I ), Width => 19 );
      NEW_LINE;
      I := I + 1;
   end loop;
end Try_Arrays;
```

INFORMATION LESSON PLAN

I. **BLOCK:** III - "Advanced Ada Topics"

II. **UNIT:** E

III. **LESSON TITLE:** "Exceptions"

IV. **LESSON OBJECTIVES:** At the completion of this lesson, the student should be able to:

1. Discuss the importance of handling exceptions.
2. Identify Ada's two types of exception type objects.
3. List and provide a brief description of the following predefined exception type objects:

   Constriant_Error
   Numeric_Error
   Storage_Error
   Program_Error
   Tasking_Error
   Status_Error
   Mode_Error
   Name_Error
   Use_Error
   Device_Error
   End_Error
   Data_Error
   Layout_Error

4. List the sequence of events which take place during the handling of an exception.

5 Provide the necessary Ada statements to declare, raise, and handle exceptions.

6. Define propagation, and understand the consequences of using handlers.

V. **LEARNING ACTIVITIES:**

1. Take notes on lecture presented by Instructor.
2. Participate in class discussion of presented lecture.
3. CAI Assignment - Block V
   AETECH *"Ada Training Environment"* or *"IntegrAda"* with *"On-Line Training and Reference Module"*.

   Read & take notes on the following sections:

   Block V, Lesson 1, Topics 1-6

   a. Exception conditions.
   b. Predefined.
   c. User defined.
   d. Handling exceptions.
   e. Propagation.
   f. Multiple exceptions.

VI. SPECIAL RESOURCES:

AETECH *"Ada Training Environment"* and *"IntegrAda"* with *"On-Line Training and Reference Module"*.

Skansholm, Ada from the Beginning, Addison-Wesley, 1988, pp. 431-445.

VII. PRESENTATION

A. Introduction

1. Discuss how even the best programs can go haywire, and how even the most thought out program needs to have a mechanism to handle unforeseen conditions.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
1. Exception Handlers	1a. Definition - An Ada structure which allows for the handling (correction) of unexpected or unforeseen circumstances, so that a program can take the appropriate action(s). 1b. Exception Declarations - Declares a name for an exception. Names are used in a) raise statements. b) exception handler. c) renaming declarations. 1c. Name 1 [, Name 2...]: exception; (can use multiple names). 1d. The response to exceptions is specified by a handler. The handler can be coded in a construct that is either a block statement, or the body of a subprogram, package, task unit, or generic unit. Such a construct is called a "frame". 1e. Handlers handle exceptions raised in their frames. 1f. Raise statement - raises an exception. raise [exception_name]; 1g. Nameless raising can occur only in a handler, and is used to re-raise the chosen exception of the handler and propagate it, even though it was handled.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
2. Predefined Exceptions	2a. Those exceptions that can be propagated by the basic operations and the predefined operators. Predefined exceptions are included on all Ada implementations. **Constraint_Error** - occurs when an attempt to violate a range constraint has been made, or to access an unknown component of a composite type. **Numeric_Error** - occurs if an attempt is made to perform an impossible numeric operation (i.e. divide by 0). **Storage_Error** - occurs if memory is exhausted. **Program_Error**- occurs upon an attempt to call a subprogram or activate a task, or elaborate a generic instantiation, when the body of the unit has not been elaborated. **Status_Error** - Occurs if an attempt is made to read from or write to a file that is not open. Also occurs if an attempt is made to open an already opened file.
2. Predefined Exceptions (continued)	**Mode_Error** - Occurs if an attempt is made to read from a file which has been opened for writing to, or attempting to write to a file which has been opened to read from. **Name_Error** - Occurs if an attempt to open a file with the wrong external file_name.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
2. Predefined Exceptions (continued)	**Use_Error** - Occurs if an attempt has been made to open a file fro an illegal use (open a LPT1: file for reading from)
	Device_Error - Occurs during a failure of an I/O device.
	End_Error - Occurs if an attempt is made to read something from a file and an EOF has been reached.
	Data_Error - Occurs when an item is read in from a file and is not of the correct type.
	Layout-Error - Occurs if an attempt is made to reference a line or column number which is beyond present boundaries.
3. User Defined Exceptions	3a. Are exceptions whose names are given in exception declaration statements. Those names can only be used in raise statement, renaming statements, and exception handlers.
4. Propagation of Exceptions	4a. Two ways to propagate an exception: a: By not handling it in the frame it occurred in. b: By using a raise statement to handle it.
	4a1. When an exception is raised, normal program execution is abandoned, and control is transferred to an exception handler.
	4a2. The selection of the handler depends on whether the exception is raised during execution of the program statements, or during elaboration of the declarations.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
4. Propagation of Exceptions (continued)	4a3. During execution of statements: Frame has a handler-control passed to handler; after a successful handling, the frame doesn't have a handler-exception propagated. 4a4. If in a subprogram body - raised at call ; If in a block-raised immediately after block; If in a package-raised at end of package frame; If in a task-task becomes complete. 4a5. If an exception occurs during execution of an exception handler, the execution of the handler is abandoned, and the above rules are followed. 4b. Exceptions should be placed at lowest program level (frames)) as possible. Each frame should handle its own unforeseen conditions. 4b1. During elaboration of declarations; if an exception is raised, then it will be propogated. If the exception was raised in a subprogram body-raised at call-abandons main; Raised in a block-raised at end of frame; Raised in a package body-raised at end of frame; Raised during a task-task completes and Tasking_Error is raised.

LABORATORY EXPERIMENT

I. **BLOCK:** III - "Advanced Ada Topics"

II. **UNIT:** E

III. **LAB NUMBER:** 27

IV. **LAB TITLE:** "Exceptions"

V. **STUDENT OBJECTIVES:** At the completion of this experiment, the student should be able to:

1. Create and use exception handlers inside subprogram units.

VI. **REQUIRED MATERIALS:**

1. Note taking materials.
2. AETECH *"IntegrAda"* with *"On-Line Training and Reference Module"*.
3. Student Data Disk.

VII. **PROCEDURE**

1. Modify the MATH.PKG created in Laboratory Experiment 16 and 17 by providing exception handlers within the package. When the driver programs prompts the user to input integer values, either for FACTORIAL, or AREA_OF_SQUARE, the exception handler should be able to handle erroneous (such as CHARACTER) input data, and prompt the user for reentry. Save your new improved package as LAB27.ADA.
2. Compile, debug, bind, and execute the program.
3. Print out a copy of your program, and your executable output to turn in to your Instructor.
4. Power down computer, and clean up area.

```
               --**********************;
               --*    Exceptions    *--;
               --**********************;

-- Author's Name          : TEACHER GUIDE ;
-- Assignment Number      : LAB # III.E ;

-----------------    Program Executive    ----------------
-- Below is a solution for Lab # III.E. This solution may
-- be used by the instructor as a guide for helping
-- students complete the laboratory assignment.
---------------------------------------------------------
with TEXT_IO, MathPkg;
use TEXT_IO, MathPkg;
procedure Triangles is

   Side1, Side2, Side3 : POSITIVE;
   Area : FLOAT;
   Valid : Boolean;
   package PositiveIO is new INTEGER_IO( POSITIVE );
   use PositiveIO;
   package FloatIO is new FLOAT_IO( FLOAT );
   use FloatIO;

begin
   put_line( "This program will calculate the area of a
   given triangle." );
   put_line( "Please enter INTEGER values when lengths are
   requested." );
   Valid := FALSE;
   while ( not Valid ) loop
    loop
     begin -- block
      NEW_LINE;
      put( "Please enter length of side one: ");
      get( Side1 );
      NEW_LINE;
      exit;
      exception
         when Data_Error =>
            put_line( "All Lengths must be Positive
            Integers!!" );
            SKIP_LINE;
         when Others =>
            put_line( "Miscellaneous Error!!" );
            raise; -- propagate
     end; -- block
    end loop;
    SKIP_LINE;
    loop
     begin -- block
      NEW_LINE;
      put( "Please enter length of side two: ");
      get( Side2 );
      NEW_LINE;
      exit;
      exception
         when Data_Error =>
            put_line( "All Lengths must be Positive
            Integers!!" );
            SKIP_LINE;
         when Others =>
            put_line( "Miscellaneous Error!!" );
            raise; -- propagate
     end; -- block
    end loop;
    SKIP_LINE;
    loop
     begin -- block
      NEW_LINE;
      put( "Please enter length of side three: ");
      get( Side3 );
      NEW_LINE;
      exit;
      exception
         when Data_Error =>
            put_line( "All Lengths must be Positive
            Integers!!" );
            SKIP_LINE;
         when Others =>
            put_line( "Miscellaneous Error!!" );
            raise; -- propagate
     end; -- block
    end loop;
    NEW_LINE;  SKIP_LINE;
  if (Side1 + Side2 > Side3) and (Side2 + Side3 > Side1)
       and (Side1 + Side3  > Side2) then
          Valid := TRUE;
          Calc_Tri_Area(Side1, Side2, Side3, Area);
    else
       put_line( "Invalid Sides!  Try Again..." );
    end if;
   end loop;
   put( "The Area of the given Triangle is: " );
   put( Area, Aft => 2, Exp => 0 );
   put_line( " Square Feet." );
   NEW_LINE;
end Triangles;
```

INFORMATION LESSON PLAN

I. **BLOCK:** III - "Advanced Ada Topics"

II. **UNIT:** F

III. **LESSON TITLE:** "Private Types"

IV. **LESSON OBJECTIVES:** At the completion of this lesson, the student should be able to:

1. Identify the uses of private and limited private types.
2. Discuss the limitations of private and limited private types.
3. Identify where private and limited private types may be declared.
4. List the three steps necessary to declare a private or limited private type.

V. **LEARNING ACTIVITIES:**

1. Take notes on lecture presented by Instructor.
2. Participate in class discussion of presented lecture.
3. CAI Assignment - Block V
   AETECH *"Ada Training Environment"* or *"IntegrAda"* with *"On-Line Training and Reference Module"*.

   Read & take notes on the following sections:

   Block V, Lesson 2, Topics 1-6.

   a. Limiting operations.
   b. Declaration.
   c. Private types.
   d. Limited private types.
   e. Hiding data structures.
   f. Information hiding.

VI. **SPECIAL RESOURCES:**

AETECH *"Ada Training Environment"* and *"IntegrAda"* with *"On-Line Training and Reference Module"*.

Skansholm, Ada from the Beginning, Addison-Wesley, 1988, pp. 372-376.

VII. PRESENTATION

A. Introduction

1. Discuss how easy it is for someone to use a package in a manner which the programmer didn't intend the package to be used. Discuss why a programmer may want to limit the availability of certain operations that the user could perform; then introduce private and limited private types.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
1. Private and Limited Private Types	1a. Are types for which the set of possible values is well defined, but not directly available to the user. This prevents user from making use of the internal structure of the type. Also, encapsulates data, where only those operations specified in the package may be performed on those type of objects.
2. Private and Limited Private Type Declarations.	2a. Are only allowed as a declarative item at the visible part of a package, or as the generic parameter declaration in a generic formal part. 2b. Limit the operations that may be performed on objects declared as private or limited private. 2c. The type declaration is in the visible part of a package; serves to limit the uses of objects of type private by outside program units. 2d. A type declaration must have a corresponding declaration of a type with the same identifier. It must appear as a declarative item of the private part of a package. 2e. Type declaration must not be an unconstrained type. 2f. Type name cannot appear within simple expressions, or in occurrences of derived types. 2g. "Private Type Declaration" creates the type corresponding; "Full Type Declaration" specifies the definition of the type.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
3. Operations of Private and Limited Private Types.	3a. Operations of a Private Type (Outside Package) - allows assignment, membership tests, selected components, qualification and explicit conversion, attributes (type and object), tests for equality or inequality. (Inside Package) - operations implicitly declared by the full type declaration. 3b. Operations of a limited Private Type - no assignment, no tests for equality or inequality; no initialization of objects, no use as a generic formal "in" parameter, no aggegrates, and no concatenation. Task type is a limited private type.
4. Declaring Private and Limited Private Types	4a. Three steps: a. Declare a type to be private or limited private. b. Identify exportable components for type. c. Complete the corresponding full type declaration in the private part of the package.

INFORMATION LESSON PLAN

I. **BLOCK:** III - "Advanced Ada Topics"

II. **UNIT:** G

III. **LESSON TITLE:** "Generics"

IV. **LESSON OBJECTIVES:** At the completion of this lesson, the student should be able to:

1. Define generic.
2. Define instantiation.
3. Instantiate a predefined generic unit.
4. List the advantages of generic type units.
5. Use generic types in a program.

V. **LEARNINIG ACTIVITIES:**

1. Take notes on lecture presented by Instructor.
2. Participate in class discussion of presented lecture.
3. CAI Assignment - Block V
   AETECH *"Ada Training Environment"* or *"IntegrAda"* with *"On-Line Training and Reference Module"*.

   Read & take notes on the following sections:

   Block V, Lesson 3, Topics 1-9

   a. Description.
   b. Generic package definition.
   c. Instantiation.
   d. Instantiation of predefined generic packages.
   e. Generic subprograms.
   f. Subprogram instantiation.
   g. Generic parameters.
   h. Passing parameters to generic packages.
   i. Generics and productivity.

VI. **SPECIAL RESOURCES:**

AETECH *"Ada Training Environment"* and *"IntegrAda"* with *"On-Line Training and Reference Module"*.

VII. PRESENTATION

A. Introduction

1. Discuss a box of generic corn flakes and describe how the box could contain Kellogg's Corn Flakes, Post Corn Flakes, etc. Then use this idea to introduce generics in programming (generic swap).

B. Instructional Topics and Key Points

TOPIC	KEY POINT
1. Generic	1a. Definition - called a template, which is parameterized or not, that allow packages and subprograms to be coded which will work for multiple types. User must provide type to generic when instantiating. 1b. Allows for units to be reused, which supports reusability, a software engineering goal. 1c. Two types of generic units, subprograms and packages. 1d. Generic declarations, along with their formal parameters, must be declared before they can be used.
2. Instantiation	2a. Definition - to create a copy of a generic package which is usable (to make a generic package available for use) by passing the required types and parameters to the package or subprogram and naming a copy it. An instance of a generic package becomes a package. An instance of a generic subprogram becomes a subprogram. 2b. Type in generic is conventionally "element". New type is passed to generic, takes the place of element type. 2c. May use either positional or named association when passing parameters to a generic. 2d. Rules for Instantiation: 1. Explicit actual for every formal unless a default. 2. Can use positional or named. 3. Expressions can match parameters of mode in.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
3. Predefined Generic Package	3a. Integer_IO Float_IO Enumeration_IO Direct_IO Sequential_IO
4. Generic Naming	4a. Outside the specifications and body of a generic unit, the name of the unit denotes the generic unit. 4b. Inside the declarative region of a generic subprogram, the name denotes the subprogram obtained by the current instantiation of the generic unit. 4c. Inside the declarative region of a package, the name denotes an instantiated package. 4d. Inside names of subprograms and packages can be overloaded, and can be recursively called.
5. Generic Formal Objects	5a. Have a mode that is either in or in out, with in as default. 5b. If declaration ends with an expression, it is the default expression (for "in").
6. Generic Formal Types	6a. Type declarations which allow an instantiation to select its types. Available types are Private, Array, Access, Discrete, Integer, Float, Fixed.
7. Generic Formal Subprograms	7a. Includes a declaration with 2 default forms: <> or (subprogram or entry(task)). 7b. Generic Bodies - are a template for the corresponding packages or subprogram bodies. Every generic subprogram must have a body. Generic bodies appear the same as bodies for non-generic units.

LABORATORY EXPERIMENT

I. **BLOCK:** III - "Advanced Ada Topics"

II. **UNIT:** G

III. **LAB NUMBER:** 28

IV. **LAB TITLE:** "Generics"

V. **STUDENT OBJECTIVES:** At the completion of this experiment, the student should be able to:

1. Write a simple generic procedure utilizing a private type.

VI. **REQUIRED MATERIALS:**

1. Note taking materials.
2. AETECH *"IntegrAda"* with *"On-Line Training and Reference Module"*.
3. Student Data Disk.

VII. **PROCEDURE**

1. Write a generic procedure which takes two objects (Object1 and Object2), and swaps their contents. Write a driver which instantiates the generic swap procedure for integer and character types. The driver should prompt the user to input an integer into Object1, input an integer into Object2, swap their contents, and display the swapped results. The same idea should be followed for the swapping of two characters. Save this program as LAB28.ADA.
2. Compile, debug, bind, and execute the program.
3. Print out a copy of your program, and your executable output to turn in to your Instructor.
4. Power down computer, and clean up area.

```
                    --******************--;
                    --*    Generics    *--;
                    --******************--;

-- Author's Name          : TEACHER GUIDE ;
-- Assignment Number      : LAB # III.G ;

-----------------   Program Executive   ---------------
-- Below is a solution for Lab # III.G. This solution may
-- be used by the instructor as a guide for helping
-- students complete the laboratory assignment.
---------------------------------------------------------
with TEXT_IO;   use TEXT_IO;

procedure Try_Generics is

  Int1, Int2 : INTEGER;
  Char1, Char2 : CHARACTER;

  generic
     type Swap_Type is private;
    procedure Swap ( Object1, Object2 : in out Swap_Type );

  procedure Swap ( Object1, Object2 : in out Swap_Type ) is
     Temp : Swap_Type := Object1;
  begin
     Object1 := Object2;
     Object2 := Temp;
  end Swap;

  procedure Swap_Ints is new Swap( INTEGER );

  procedure Swap_Chars is new Swap( CHARACTER );

  package IntegerIO is new INTEGER_IO( INTEGER );
  use IntegerIO;

begin -- Try_Generics
  put( "Please enter an Integer value: " );
  get( Int1 );
  NEW_LINE;
  put( "Now enter a Second Integer value: " );
  get( Int2 );
  NEW_LINE;   NEW_LINE;
  Swap_Ints( Int1, Int2 );
  put_line( "After Swapping Values: " );
  put( "      First Integer is: ===> " );
  put( Int1, Width => 1 );
  NEW_LINE;
  put( "And Second Integer is: ===> " );
  put( Int2, Width => 1 );
  NEW_LINE;  NEW_LINE;  NEW_LINE;
  put( "Please enter one Character: " );
  get( Char1 );
  NEW_LINE;
  put( "Now enter a Second Character: " );
  get( Char2 );
  NEW_LINE;  NEW_LINE;
  Swap_Chars( Char1, Char2 );
  put_line( "After Swapping Values: " );
  put( "     First Character is: ===> " );
  put( Char1 );
  NEW_LINE;
  put( "And Second Character is: ===> " );
  put( Char2 );
  NEW_LINE;
end Try_Generics;
```

INFORMATION LESSON PLAN

I. **BLOCK:** III - "Advanced Ada Topics"

II. **UNIT:** H

III. **LESSON TITLE:** "Sequential Files"

IV. **LESSON OBJECTIVES:** At the completion of this lesson, the student should be able to:

1. Define sequential file.
2. Create a sequential file.
3. Open, close, read, and write sequential files and their associated information.
4. Identify the following file functions:

Mode	**Name**	**Form**
Is_Open	**End_Of_File**	

V. **LEARNING ACTIVITIES:**

1. Take notes on lecture presented by Instructor.
2. Participate in class discussion of presented lecture.
3. CAI Assignment - Block VI
   AETECH *"Ada Training Environment"* or *"IntegrAda"* with *"On-Line Training and Reference Module"*.

   Read & take notes on the following sections:

   Block VI, Lesson 2, Topics 1-8, 10

   a. Packages.
   b. Nontextual data.
   c. File objects
   d. File modes.
   e. Creating and opening files.
   f. Closing, resetting, and deleting files.
   g. Instantiation.
   h. Sequential_IO reading and writing.
   i. Useful file functions.

VI. **SPECIAL RESOURCES:**

AETECH *"Ada Training Environment"* and *"IntegrAda"* with *"On-Line Training and Reference Module"*.

Skansholm, Ada From The Beginning, Addison-Wesley, 1988, pp. 492-498.

VII. PRESENTATION

A. Introduction

1. Choose a file from a convenient filing cabinet, and discuss this file , why it is a file , and what information a file can contain; then introduce sequential files.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
1. Sequential File	1a. Definition - A group of related information whose access is somewhat limited by having to read, or write information in a sequential (from first to last) manner. The file is viewed as a sequence of values that are transferred in the order of their appearance, as produced by the program or by the environment.
2. Sequential File Operations Note: (P) = procedure (F) = function	2a. (P)Create - Gives a name to operating system storage device from a previously declared file object (My_file:File_type). Establishes a new external file with the given name and form, and associates this external file with the given internal name. Assigns the file object a file mode. Default mode is Out_File. 2b. (P)Read - Reads information from a previously opened file in sequential order. Reads an element from a given file, and returns the value of the element in the item parameter. 2c. (P)Write -writes information to a previously created file, in sequential order. Writes the value of item to the given file.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
2. Sequential Files Operations (continued)	2d. (P)Close -Closes a previously opened file. Severs the association between the given file and its external file. The file is left closed. 2e. Reset - Resets file pointer of a sequential file to the first element in the file. 2f. (P)Open - Associates the given file with an existing external file, and sets the mode of the file. The given file is left open. 2g. (P)Delete - Deletes the external file associated with the given internal name. The given file is closed, and the external file ceases to exist.
3. File Functions	3a. Mode - returns current mode of the given file. 3b. Name - Returns a string which identifies the external file. 3c. IS_OPEN - Returns True if the file is open, otherwise returns false. 3d. End_of_file - Operates on a file of mode In_ File. Returns True if no more elements can be read from the given file; otherwise, it returns False. 3e. Form - Identifies the file's properties (i.e."save for 90 days").
4. Instantiation	4. Because Sequential_IO is a generic package, it must be instantiated for a given data type, using parameter Element_Type. (i.e. package Int_IO is new Sequential_IO (Integer):
5. Conventional Naming Techniques	5. Use identifiers which are imperative verbs or nouns. Imperative verbs name actions, nouns name values or conditions. This naming convention is used in Sequential_IO, where nouns are function names, and imperative verbs are procedure names.

LABORATORY EXPERIMENT

I. **BLOCK:** III - "Advanced Ada Topics"

II. **UNIT:** H

III. **LAB NUMBER:** 29

IV. **LAB TITLE:** "Sequential Files"

V. **STUDENT OBJECTIVES:** At the completion of this experiment, the student should be able to:

1. Create a sequential file.
2. Instantiate the generic package Sequential_IO.
3. Write information to the created sequential file.
4. Read previously stored information from the sequential file.
5. Close the sequential file.

VI. **REQUIRED MATERIALS:**

1. Note taking materials.
2. AETECH *"IntegrAda"* with *"On-Line Training and Reference Module*.
3. IntegrAda Environment or Alsys "AdaUser" Libraries.
3. Student Data Disk.

VII. **PROCEDURE**

1. Create a sequential file to handle the input and output of data of the following type:

```
type PersonDataType is record
     Name:String(1..10);
     Age :Integer;
     Favorite_Color:COLORS.A_COLOR;
end record;
```

The sequential file should provide input and output for the following data:

Name	Age	Favorite Color
1. Susie	23	WHITE
2. Fred	12	RED
3. Barney	10	BLUE
4. Debbie	24	MAGENTA
5. Sam	18	GREEN
6. Andy	16	YELLOW
7. Amy	18	BLUE

Create a procedure to write the above data into a sequential file, from the keyboard. Save this program as LAB29A.ADA.

2. Create a procedure which will read the data from the sequential file, and display the data to the screen as shown above. Save this program as LAB29B.ADA.

3. Compile, debug, bind, and execute the programs.

4. Print out a copy of each program, and a copy of your executable output to turn in to your Instructor.

5. Write a procedure to display only the fourth name from the list above, and that name's age and favorite color. Save this program as LAB29C.ADA. Print out a copy of your program, and executable code to turn in to your Instructor.

6. Power down computer, and clean up area.

```
--***************************--;
--*    Sequential Files    *--;
--***************************--;

-- Author's Name          : TEACHER GUIDE ;
-- Assignment Number      : LAB # III.H ;

-----------------   Program Executive   ----------------
-- Below is a solution for Lab # III.H. This solution may
-- be used by the instructor as a guide for helping
-- students complete the laboratory assignment.
---------------------------------------------------------
with TEXT_IO, SEQUENTIAL_IO, Colors;
use TEXT_IO, Colors;

procedure Seq_Write is

   type Person_Data is
         record
           Name           : STRING( 1 .. 10 );
           Age            : INTEGER;
           Favorite_Color : Colors.A_Color;
         end record;

   package IntegerIO is new INTEGER_IO( INTEGER );

   package ColorIO is new ENUMERATION_IO( Colors.A_Color);
   use ColorIO;

   package PersonIO is new SEQUENTIAL_IO( Person_Data );
   use PersonIO;

   Temp_Name : STRING( 1 .. 10 );
   I : NATURAL;

   Person_File : PersonIO.FILE_TYPE;

   Person : Person_Data;

begin
   create( file => Person_File, name => "People.DAT" );
      -- uses default (and required) mode value of
OUT_FILE

   put( "Enter Name ( type END to quit ): ===> " );
   I := 0;
   while ( not End_of_Line ) loop
      I := I + 1;
      get( Temp_Name( I ) );
   end loop;
   Temp_Name( I + 1 .. 10 ) := ( I + 1 .. 10 => ' ' );
   NEW_LINE;
   while ( Temp_Name /= "END       " ) loop
      Person.Name := Temp_Name;
      put( "Enter Person's Age: =============> " );
      IntegerIO.get( Person.Age );
      NEW_LINE;
      put( "Enter Person's Favorite Color: ==> " );

      ColorIO.get( Person.Favorite_Color );

      NEW_LINE;  NEW_LINE;
      SKIP_LINE;

      write( file => Person_File, item => Person );

      put( "Enter Name ( type END to quit ): ===> " );
      I := 0;
      while ( not End_of_Line ) loop
         I := I + 1;
         get( Temp_Name( I ) );
      end loop;
      Temp_Name( I + 1 .. 10 ) := ( I + 1 .. 10 => ' ' );
      NEW_LINE;
   end loop;

   close( file => Person_File );

end Seq_Write;
```

```
with TEXT_IO, SEQUENTIAL_IO, Colors;
use TEXT_IO, Colors;

procedure Seq_Read is

   type Person_Data is
         record
           Name           : STRING( 1 .. 10 );
           Age            : INTEGER;
           Favorite_Color : Colors.A_Color;
         end record;

   package IntegerIO is new INTEGER_IO( INTEGER );

   package ColorIO is new ENUMERATION_IO( Colors.A_Color);
   use ColorIO;

   package PersonIO is new SEQUENTIAL_IO( Person_Data );
   use PersonIO;
   Person_File : PersonIO.FILE_TYPE;
   Person : Person_Data;
   I : INTEGER;

begin
  put_line( "    NAME          AGE       FAVORITE COLOR");
   open( file => Person_File, mode => in_file, name =>
"People.DAT" );
   I := 0;
   while  ( not End_Of_File( Person_File ) )  loop
      I := I + 1;
      read( file => Person_File, item => Person );
      IntegerIO.put( I, Width => 1 );
      put( ".  " );
      put( Person.Name );
      IntegerIO.put( Person.Age, Width => 9 );
      put( "           ");
      ColorIO.put( Person.Favorite_Color );
      NEW_LINE;
   end loop;

   close( file => Person_File );

end Seq_Read;
```

```
with TEXT_IO, SEQUENTIAL_IO, Colors;
use TEXT_IO, Colors;
procedure Seq_Rd_4 is

   type Person_Data is
         record
           Name           : STRING( 1 .. 10 );
           Age            : INTEGER;
           Favorite_Color : Colors.A_Color;
         end record;

   package IntegerIO is new INTEGER_IO( INTEGER );

   package ColorIO is new ENUMERATION_IO( Colors.A_Color);
   use ColorIO;

   package PersonIO is new SEQUENTIAL_IO( Person_Data );
   use PersonIO;

   Person_File : PersonIO.FILE_TYPE;
   Person : Person_Data;
   I : POSITIVE;

begin
```

```
   open( file => Person_File, mode => in_file, name =>
"People.DAT" );

   for  I in 1 .. 4  loop
      read( file => Person_File, item => Person );
   end loop;
   NEW_LINE;
   put( "Fourth Person's Name: ============>  " );
   put( Person.Name );
   NEW_LINE;
   put( "Fourth Person's Age: =============>  " );
   IntegerIO.put( Person.Age, Width => 1 );
   NEW_LINE;
   put( "Fourth Person's Favorite Color: ==>  " );
   ColorIO.put( Person.Favorite_Color );
   NEW_LINE;

   close( file => Person_File );

end Seq_Rd_4;
```

INFORMATION LESSON PLAN

I. **BLOCK:** III - "Advanced Ada Topics"

II. **UNIT:** I

III. **LESSON TITLE:** "Direct Access Files"

IV. **LESSON OBJECTIVES:** At the completion of this lesson, the student should be able to:

1. Define direct access file.
2. Create a direct access file.
3. Open, close, read, and write direct access files and their associated information.
4. Identify the file functions Size, Index and Set_Index which allow a user to directly access a particular file item.

V. **LEARNING ACTIVITIES:**

1. Take notes on lecture presented by Instructor.
2. Participate in class discussion of presented lecture.
3. CAI Assignment - Block VI
   AETECH *"Ada Training Environment"* or *"IntegrAda"* with *"On-Line Training and Reference Module"*.

   Read & take notes on the following sections:

   Block VI, Lesson 2, Topics 1-7, 9, 10.

   a. Packages.
   b. Nontextual data.
   c. File objects.
   d. File modes.
   e. Creating and opening files.
   f. Closing, resetting, and deleting files.
   g. Instantiation.
   h. Direct_IO reading and writing.
   i. Useful_file functions.

VI. SPECIAL RESOURCES:

AETECH *"Ada Training Environment"* and *"IntegrAda"* with *"On-Line Training and Reference Module"*.

Skansholm, Ada From The Beginning, Addison-Wesley, 1988, pp. 513-518.

VII. PRESENTATION

A. Introduction

1. Explain that accessing record 999,999 in a file containing a million records would take an enormous amount of time; then introduce direct access files, which allow a user to go directly to the required record.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
1. Direct Access Files	1a. Definition - A file containing a list of index numbers which allows users to directly access the record within the file by use of its index number. For direct access, the file is viewed as a set of elements occupying consecutive positions in linear order: a value can be transferred to or from an element of the file at any selected position.
2. Direct File Operations	2a. Open and close same as sequential files. An open file has a current mode, which is a value of one of the mode types. 2b. Read and write use additional parameter "from" which is equivalent to index number.
3. File Functions	3a. Mode, Name, From, Is_Open, End_Of_File same as sequential files. 3b. Size - Returns number of items in file (number of index #'s). Operates on a file of any mode. 3c. (P)Index - Operates on a file of any mode, returns the current index of a given file. 3d. (P)Set_Index - Operates on a file of any mode. Sets the current index of the given file to the given value (which may exceed the current size of the file).

B. Instructional Topics and Key Points

TOPIC	KEY POINT
4. Instantiation	4a. Direct_IO is a generic package; therefore it must be instantiated with a given type, name information, and given a new name. The resulting package contains the declaration of a file type (called File_Type) for sets of elements (of the given type) as well as the operations applicable to these files (open, reset, etc.).

LABORATORY EXPERIMENT

I. **BLOCK:** III - "Advanced Ada Topics"

II. **UNIT:** I

III. **LAB NUMBER:** 30

IV. **LAB TITLE:** "Direct Access Files"

V. **STUDENT OBJECTIVES:** At the completion of this experiment, the student should be able to:

1. Create Ada procedures to write and read direct access files.

VI. **REQUIRED MATERIALS:**

1. Note taking materials.
2. AETECH *"IntegrAda"* with *"On-Line Training and Reference Module"*.
3. Student Data Disk.

VII. **PROCEDURE**

1. Using the data provided in Lab 29, create a procedure which stores the given data in a direct access file. Save this program as LAB30A.ADA.
2. Create a procedure which will read the fourth name from the given file, along with the name's age and favorite color. Save this program as LAB30B.ADA.
3. Compile, debug, bind, and execute the programs.
4. Print out a copy of each program, and a copy of your executable output to turn in to your Instructor.
5. Power down computer, and clean up area.

```
               --*****************************--;
               --*    Direct Access Files    *--;
               --*****************************--;

-- Author's Name          : TEACHER GUIDE ;
-- Assignment Number      : LAB # III.I ;

-----------------    Program Executive    ---------------
-- Below is a solution for Lab # III.I. This solution may
-- be used by the instructor as a guide for helping
-- students complete the laboratory assignment.
---------------------------------------------------------
with TEXT_IO, DIRECT_IO, Colors;
use TEXT_IO, Colors;

procedure Direct_Write is

   type Person_Data is
         record
           Name           : STRING( 1 .. 10 );
           Age            : INTEGER;
           Favorite_Color : Colors.A_Color;
         end record;

   package IntegerIO is new INTEGER_IO( INTEGER );

   package ColorIO is new ENUMERATION_IO( Colors.A_Color);
   use ColorIO;

   package PersonIO is new DIRECT_IO( Person_Data );
   use PersonIO;

   Temp_Name : STRING( 1 .. 10 );
   I : NATURAL;

   Person_File : PersonIO.FILE_TYPE;

   Person : Person_Data;

begin
   create( file => Person_File, mode => INOUT_FILE, name =>
"Persons.DAT" );

   put( "Enter Name ( type END to quit ): ===> " );
   I := 0;
   while ( not End_of_Line ) loop
      I := I + 1;
      get( Temp_Name( I ) );
   end loop;
   Temp_Name( I + 1 .. 10 ) := ( I + 1 .. 10 => ' ' );
   NEW_LINE;
   while ( Temp_Name /= "END       " ) loop
```

```
      Person.Name := Temp_Name;
      put( "Enter Person's Age: =============> " );
      IntegerIO.get( Person.Age );
      NEW_LINE;
      put( "Enter Person's Favorite Color: ==> " );
         -- might want to list available colors.
      ColorIO.get( Person.Favorite_Color );
      NEW_LINE;  NEW_LINE;
      SKIP_LINE;

      write( file => Person_File, item => Person );

      put( "Enter Name ( type END to quit ): ===> " );
      I := 0;
      while ( not End_of_Line ) loop
         I := I + 1;
         get( Temp_Name( I ) );
      end loop;
      Temp_Name( I + 1 .. 10 ) := ( I + 1 .. 10 => ' ' );
      NEW_LINE;
   end loop;
   close( file => Person_File );

end Direct_Write;
```

```
with TEXT_IO, DIRECT_IO, Colors;
use TEXT_IO, Colors;

procedure Dir_Rd_4 is

   type Person_Data is
         record
           Name           : STRING( 1 .. 10 );
           Age            : INTEGER;
           Favorite_Color : Colors.A_Color;
         end record;

   package IntegerIO is new INTEGER_IO( INTEGER );

   package ColorIO is new ENUMERATION_IO( Colors.A_Color);
   use ColorIO;

   package PersonIO is new DIRECT_IO( Person_Data );
   use PersonIO;

   Person_File : PersonIO.FILE_TYPE;
   Person      : Person_Data;

begin
   open( file => Person_File, mode => in_file, name =>
         "Persons.DAT" );
   read( file => Person_File, item => Person, from => 4 );

   close( file => Person_File );

   NEW_LINE;
   put( "Fourth Person's Name: ============>  " );
   put( Person.Name );
   NEW_LINE;
   put( "Fourth Person's Age: =============>  " );
   IntegerIO.put( Person.Age, Width => 1 );
   NEW_LINE;
   put( "Fourth Person's Favorite Color: ==>  " );
   ColorIO.put( Person.Favorite_Color );
   NEW_LINE;

end Dir_Rd_4;
```

INFORMATION LESSON PLAN

I. **BLOCK:** III - "Advanced Ada Topics"

II. **UNIT:** J

III. **LESSON TITLE:** "Introduction to Tasks"

IV. **LESSON OBJECTIVES:** At the completion of this lesson, the student should be able to:

1. Define task types and objects.
2. Identify the two parts of a task programming unit.
3. Declare task types and objects.
4. Understand task compilation.
5. Understand how a task is started.
6. Understand how a task ends.

V. **LEARNING ACTIVITIES:**

1. Take notes on lecture presented by Instructor.
2. Participate in class discussion of presented lecture.
3. CAI Assignment - Block IV
   AETECH *"Ada Training Environment"* or *"IntegrAda"* with *"On-Line Training and Reference Module"*.

   Read & take notes on the following sections:

   Block IV, Lesson 3, Topics 1-7

   a. Tasking in embedded computers.
   b. Structure of a task.
   c. Task types.
   d. Encapsulating tasks.
   e. Separate compilation.
   f. Starting tasks.
   g. Ending a task.

VI. **SPECIAL RESOURCES:**

AETECH *"Ada Training Environment"* and *"IntegrAda"* with *"On-Line Training and Reference Module"*.

VII. PRESENTATION

A. Introduction

1. Describe the cockpit of an airplane with many computers in it, all working at the same time to keep the airplane functioning. Describe the need for these computers to communicate with one another to ensure that the plane is operating correctly; then introduce tasks.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
1. Task	1a. Tasks are program units whose executions proceed in parallel; may use different processors, and would synchronize their execution in order to process data.
	1b. The properties of a task are defined in its specification and body. Specifications are the interface, and bodies are the executable statements.
	1c. Specifications that begin with the reserved word **task** type declare a type of task. Objects may then be declared of that type. Specifications which begin with only the reserved word **task** declare a single task object of an anonymous type.
	1d. Tasks may not be compiled alone; they must be included in a declarative part of a structure (i.e. subprogram, package body, block, etc.).
	1e. Task specifications and bodies are Ada compilation units, and as such, may be compiled separately from one another.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
2. Running Tasks	2a. If multiple task objects are declared in the declarative region of a program unit, activation occurs after passing the reserved word **begin** of the unit. If they are in a package, activation occurs after the declarative part of the package body is elaborated.
	2b. Each task depends on at least one "master". A "master" can be a task, block, subprogram, or package. Masters complete when their **end** statement is reached; unless they have dependent tasks, then they complete only when no dependents are left active.
	2c. Task types are considered **limited private** types; hence, neither comparison (=,/=) or assignment (:=) are available for objects of **task type**.
	2d. Tasks are considered frames; hence, they can have exception handlers. Exceptions that occur during task activation complete the task and then raise **Tasking_Error** in the declarative region they are being activated in.

LABORATORY EXPERIMENT

I. **BLOCK:** III - "Advanced Ada Topics"

II. **UNIT:** J

III. **LAB NUMBER:** 31

IV. **LAB TITLE:** "Introduction to Tasks"

V. **STUDENT OBJECTIVES:** At the completion of this experiment, the student should be able to:

1. Write a procedure with an internal task which runs concurrently with the procedure until a <CTRL C> is pressed.

VI. **REQUIRED MATERIALS:**

1. Note taking materials.
2. AETECH *"IntegrAda"* with *"On-Line Training and Reference Module"*.
3. Student Data Disk.

VII. **PROCEDURE**

1. Write a procedure which outputs to the screen "This is the procedure". Make this procedure an infinite loop. Include in the procedure a task which outputs to the screen "This is the task". Make the task an infinite loop. Program execution is terminated when <CTRL C> is pressed. Save your program as LAB31.ADA.
2. Compile, debug, bind, and execute the program.
3. Print out a copy of your program, and your executable code to turn in to your Instructor.
4. Power down computer, and clean up area.

```
              --*******************************--;
              --*    Introduction to Tasks    *--;
              --*******************************--;

-- Author's Name          : TEACHER GUIDE ;
-- Assignment Number      : LAB # III.J ;

----------------    Program Executive    ----------------
-- Below is a solution for Lab # III.J. This solution may
-- be used by the instructor as a guide for helping
-- students complete the laboratory assignment.
---------------------------------------------------------
with TEXT_IO;   use TEXT_IO;

procedure Task_Demo is

   task Print_Msg;

   task body Print_Msg is
   begin
      loop
         put_line( "This is the task." );
      end loop;
   end Print_Msg;

begin
   loop
      put_line( "This is the procedure." );
   end loop;
end Task_Demo;
```

INFORMATION LESSON PLAN

I. **BLOCK:** III - " Advanced Ada Topics"

II. **UNIT:** K

III. **LESSON TITLE:** "Tasks and Task Communication"

IV. **LESSON OBJECTIVES:** At the completion of this lesson, the student should be able to:

1. Describe how tasks communicate.
2. Define rendezvous.
3. Describe how a block of action within a task can be performed through an accept..do structure.
4. Discuss the use of the select statement.
5. List the two ways a task can end.
6. Define the following task attributes:

   'Callable   'Terminated   'Storage_Size

V. **LEARNING ACTIVITIES:**

1. Take notes on lecture presented by Instructor.
2. Participate in class discussion of presented lecture.
3. CAI Assignment - Block IV
   AETECH *"Ada Training Environment"* or *"IntegrAda"* with *"On-Line Training and Reference Module"*.

   Read & take notes on the following sections:

   Block IV, Lesson 3, Topics 8-15.

   a. Communication in tasks.
   b. Rendezvous.
   c. Accept and do.
   d. The select statement.
   e. Receive statement.
   f. Termination.
   g. Abort versus terminate.
   h. Task units.

VI. SPECIAL RESOURCES:

AETECH *"Ada Training Environment"* and *"IntegrAda"* with *"On-Line Training and Reference Module"*.

VII. PRESENTATION

A. Introduction

1. Discuss the importance of tasks being able to communicate with one another (cockpit example), rather than just merely continuing to execute on their own.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
1. Task Communication	1a. Tasks can have entries (specified in their specifications). An entry of a task can be called (by name) by other units. The called task executes an accept statement (in its body) for the entry, and "accepts" the call. Synchronization is the rendezvous between an entry call and an accept. Since entries can have parameters (i.e. data to share), synchronization provides the basic means for communication between tasks. 1b. Entry Calls- execution begins with evaluation of name, parameters; then if an accept statement to the call has been reached, the call is received. If the accept statement has not been reached, the call is suspended, and multiple waiting calls are queued.
2. Rendezvous	2a. Definition - When two tasks meet together through an entry and accept. Once rendezvous is complete, tasks resume independent operation.

B. Instructional Topics and Key points

TOPIC	KEY POINT
3. Accept..Do Structure	3a. Execution of a delay statement evaluates the simple expression, and suspends execution for at least the duration specified by the result of the expression. The expression must be of predefined type Duration with range 0..86400 seconds (one day). 3b. Predefined package Calendar provides time resources (i.e. function Clock, type Time, etc.).
4. Select Statements	4a. Used to control task. Three forms: 1. Selective waits allows selecting from one or more alternatives. Must have at least one accept alternative. Can have only one of the following: terminate, else, delay. An alternative is said to be open if it has no "when" or if the condition following the "when" is true, otherwise it is closed. If an alternative is closed and there is no "else" part, tasks can wait until an alternative is selected. An open delay is selected if no other open can be selected before the specified time. An open terminate can only be selected if all entries are ended. 2. Conditional Entry Calls Issues an entry call, if a rendezvous is not immediately available, cancels the entry call (does else part). 3. Timed Entry Calls -Issues an entry call, if a rendezvous is not started within the given delay, the call is cancelled.

B. Instructional Topics and Key Points

TOPIC	KEY POINT
5. Task Attributes (where T is task object, and E is entry of task T)	5a. T 'Callable - Returns true if task is not terminated. Returns False if T is completed, terminated, or abnormal. T 'Terminated - Returns True if task has been called and has completed; returns False otherwise. T 'Storage_Size returns size of memory allocated for task. E 'Count - returns number of entry calls queued on entry E.

LABORATORY EXPERIMENT

I. **BLOCK:** III - "Advanced Ada Topics"

II. **UNIT:** K

III. **LAB NUMBER:** 32

IV. **LAB TITLE:** "Task Communication"

V. **STUDENT OBJECTIVES:** At the completion of this experiment, the student should be able to:

1. Write a procedure in which two tasks communicate with a main procedure.

VI. **REQUIRED MATERIALS:**

1. Note taking materials.
2. AETECH *"IntegrAda"* with *"On-Line Training and Reference Module"*.
3. Student Data Disk.

VII. **PROCEDURE**

1. Write a procedure which declares two tasks. The procedure shall prompt the user to input either a 1 or a 2. If 1 is input, task1 is communicated with and outputs to the screen "Task 1 communication complete". If the user enters 2, then task2 is communicated with, and outputs to the screen "Task 2 communication complete". Any other input other than 1 or 2 terminates both tasks, and the procedure. Utilize a case structure. Save your program as LAB32.ADA.
2. Compile, debug, bind, and execute the program.
3. Print out a copy of your program and executable code to turn in to your Instructor.
4. Power down computer, and clean up area.

```
                   --*****************************--;
                   --*    Task Communication     *--;
                   --*****************************--;

-- Author's Name               : TEACHER GUIDE ;
-- Assignment Number           : LAB # III.K ;

----------------    Program Executive    ----------------
-- Below is a solution for Lab # III.K. This solution may
-- be used by the instructor as a guide for helping
-- students complete the laboratory assignment.
---------------------------------------------------------
with TEXT_IO;   use TEXT_IO;

procedure Task_Demo_2 is

   Response : CHARACTER;
   Done : BOOLEAN;

   task One is
      entry Print;
      entry Quit;
   end One;

   task Two is
      entry Write;
      entry Leave;
   end Two;

   task body One is
      OKToQuit : BOOLEAN;
   begin
      OKToQuit := FALSE;
      while ( NOT OKToQuit ) loop
         select
           accept Print do
              put_line( "Task 1 communication complete." );
           end Print;
         or
           accept Quit do
              OKToQuit := TRUE;
           end Quit;
         end select;
      end loop;
   end One;

   task body Two is
      OKToLeave : BOOLEAN;
   begin
      OKToLeave := FALSE;
      while ( NOT OKToLeave ) loop
         select
```

```
           accept Write do
              put_line( "Task 2 communication complete." );
           end Write;
         or
           accept Leave do
              OKToLeave := TRUE;
           end Leave;
         end select;
      end loop;
   end Two;

begin
  Done := FALSE;
  while ( NOT Done ) loop
    put_line("Enter a 1 to communicate with Task One...");
    put_line("a 2 to communicate with Task Two...");
    put_line("or anything else to terminate..." );
    NEW_LINE;
    put("Please enter your choice now ===> ");
    get( Response );
    NEW_LINE;  NEW_LINE;

    case Response is
       when '1' => One.Print;
       when '2' => Two.Write;
       when others =>
          One.Quit;
          Two.Leave;
          Done := TRUE;
    end case;

    NEW_LINE;  NEW_LINE;
 end loop;
end Task_Demo_2;
```